

**NEUE METHODEN
DER BEHANDLUNG VON SYNTAXFEHLERN
BEI EINFACHEN PRÄZEDENZSPRACHEN**

Karl-Rudolf Moll

1970

NEUE METHODEN DER BEHANDLUNG VON SYNTAXFEHLERN
BEI EINFACHEN PRÄZEDENZSPRACHEN

D I S S E R T A T I O N

Zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
des Fachbereichs Mathematik
der Ludwig-Maximilians-Universität
München



vorgelegt von Karl-Rudolf Moll

aus München

eingereicht im Dezember 1977



1. Referent: Prof. Dr. G. Seegmüller

Korreferent: Prof. Dr. G. Hämmerlin

Tag der mündlichen Prüfung: 12. 7. 1978

DISSERTATIONS- UND FOTODRUCK FRANK oHG
8 München 2, Gabelsbergerstraße 15, Tel. 2 8 0 9 0 9 0

Herrn Professor Dr. G. Seegmüller danke ich dafür, daß er die Auseinandersetzung mit dem Problem der Behandlung von Syntaxfehlern angeregt und unterstützt, sowie durch vielseitigen Rat stets gefördert hat.

Meinem Kollegen, Herrn P. Gonser danke ich dafür, daß er stets als sachkundiger Diskussionspartner zur Verfügung stand.

Vielen Mitarbeitern des Leibniz-Rechenzentrums der Bayerischen Akademie der Wissenschaften habe ich für so manche temporäre Entlastung von anderen Aufgaben zu danken.

INHALTSVERZEICHNIS

0.	Einführung, Übersicht und Organisation	1
0.1	Einführung in die Problematik der Behandlung von Syntaxfehlern	1
0.2	Abschnittsübersicht	6
0.3	Organisatorische Bemerkungen	7
1.	Grundlagen	8
1.1	Grundbegriffe aus der naiven Mengenlehre	8
1.1.1	Mengen	8
1.1.2	Relationen	10
1.1.3	Abbildungen	11
1.2	Grundbegriffe aus der Theorie der Sprachübersetzung	12
1.2.1	Formale Sprachen	12
1.2.2	Sprachdefinition mit Hilfe von kontextfreien Grammatiken	13
1.2.3	Einfache Präzedenzsprachen	22
1.3	Die Beschreibung von Analysealgorithmen für einfache Präzedenzsprachen	26
1.3.1	Anschauliche Beschreibung	26
1.3.2	Beschreibung mit Hilfe eines Programmes	29
1.3.3	Formale Beschreibung mit Hilfe von Analyse- funktionen	31
2.	Die Erkennung von Syntaxfehlern mit Hilfe von Analysatoren für einfache Präzedenzsprachen	42
2.1	Zur frühestmöglichen Erkennung von Syntaxfehlern	42

2.2	Die Fehlererkennung der bisher bekannten Analysatoren	43
2.2.1	Die Fehlererkennung des Analysators von Wirth-Weber	43
2.2.2	Die Fehlererkennung des Analysators von Leinius	46
2.2.3	Die Fehlererkennung des Analysators von Rhodes	48
2.3	Entwurf eines Analysators für einfache Präzedenzsprachen, welche die Existenz von Syntaxfehlern frühestmöglich er- kennt	52
2.3.1	Anschauliche Darstellung aller Fälle, in denen der Algorithmus von Rhodes Fehler nicht frühestmöglich erkennt	52
2.3.2	Die Menge der gesuchten Variablen	55
2.3.3	Ein Analysator, welcher Syntaxfehler frühestmöglich erkennt	69
3.	Neue Methoden der Behandlung von Syntaxfehlern bei einfachen Präzedenzsprachen	72
3.1	Der Fehlerbegriff als Grundlage für die Diskussion von fehlerbehandelnden Analysealgorithmen	72
3.2	Grundzüge und Diskussion der bisher bekannten Methoden	80
3.2.1	Die Methode von Wirth	81
3.2.2	Die Methode von Leinius	85
3.2.3	Die Methode von Rhodes	89
3.2.4	Zusammenfassende Wertung	93
3.3	Nachsilbenanalyse als Fehlerbehandlung für einfache Präze- denzsprachen	95
3.3.1	Beschreibung und Diskussion des Prinzips der Nachsilbenanalyse	95
3.3.2	Theoretische Grundlagen für die Realisierung von fehlerbehandelnden Analysealgorithmen, welche Nach- silbenanalysen produzieren	100

3.3.3	Nachsilbenanalysatoren für einfache Präzedenz- sprachen	105
3.3.3.1	Anschauliche Beschreibung der Kon- struktion der Analysatoren	105
3.3.3.2	Formale Beschreibung der Analysatoren für Nachsilben	113
3.3.3.2.1	Analysefunktionen für Nachsilbensprachen	114
3.3.3.2.2	Erweiterung der Analysefunktionen für einfache Präzedenzsprachen zu Analyse- funktionen für deren Nachsilbensprachen	116
4.	Übertragung der bisher gewonnenen Ergebnisse auf Analysatoren, welche auf Präzedenzfunktionen beruhen	138
4.1	Analyse mit Hilfe von Präzedenzfunktionen	138
4.2	Übertragung der Fehlererkennung	143
4.3	Übertragung der Fehlerbehandlung	144
5.	Zusammenfassung	145
6.	Literaturverzeichnis	147
7.	Index	155
7.1	Symbole	155
7.2	Spezielle Mengen	158
7.3	Spezielle Relationen	160
7.4	Spezielle Abbildungen	161
7.5	Wichtige Begriffe	163
7.6	Definitionen, Sätze, Hilfssätze, Bemerkungen, Beispiele, Bilder	167

0. EINFÜHRUNG, ÜBERSICHT UND ORGANISATION

0.1 Einführung in die Problematik der Behandlung von Syntaxfehlern

Die in Lehrbüchern (z.B. [Gri,71]) üblicherweise beschriebenen Syntaxanalysealgorithmen für spezielle Klassen von Chomsky-2 Sprachen beenden die Analyse einer fehlerhaften Eingabezeichenfolge, wenn sie erkannt haben, daß der bisher gelesene Teil der Eingabezeichenfolge kein Präfix eines Wortes der zu analysierenden Sprache ist. Für Übersetzer, welche in der Praxis benützt werden, sind solche Syntaxanalysealgorithmen unbrauchbar, da bei jeder Übersetzung eines Programms nur ein syntaktischer Fehler erkannt wird. Deswegen beschäftigen sich zahlreiche Veröffentlichungen der letzten Jahre mit dem Problem, diese Analysealgorithmen so zu erweitern, daß mehr als ein Syntaxfehler erkannt wird. Von einem um eine Fehlerbehandlung erweiterten Analysealgorithmus erwartet man die folgenden Eigenschaften:

- E1. Jede Eingabezeichenfolge muß vollständig analysiert werden. Durch die Korrektur der gemeldeten Fehler muß eine fehlerfreie Zeichenfolge entstehen. Diese sollte der vom Benutzer beabsichtigten möglichst ähnlich sein.
- E2. Der Fehlerbehandlungsalgorithmus darf nur von der Grammatikklasse, nicht aber von der Sprache selbst abhängen, d.h. er muß automatisch erzeugbar sein.
- E3. Der Zeit- und Platzbedarf des Analysealgorithmus für die Analyse von fehlerhaften Zeichenfolgen, gebildet aus gängigen Sprachelementen, muß in derselben Potenz von der Länge der Eingabezeichenfolge abhängen, wie für die Analyse von fehlerfreien Zeichenfolgen.
- E4. Die Effizienz der Analyse fehlerfreier Eingabezeichenfolgen darf durch die Vorkehrungen für eine eventuelle Fehlerbehandlung nicht wesentlich verschlechtert werden.

Bei der Realisierung der Eigenschaft E1 tritt folgende Schwierigkeit auf: Dem fehlerbehandelnden Algorithmus ist nur die fehlerhafte Zeichenfolge, nicht aber die vom Programmierer beabsichtigte bekannt. Das heißt,

der fehlerbehandelnde Algorithmus kann nur Annahmen über die Art der syntaktischen Fehler machen, absolute Aussagen sind nicht möglich. Damit die Fehlermeldungen für den Programmierer brauchbar sind, ist es notwendig, daß die gemeldeten Fehler in möglichst vielen Fällen mit den tatsächlichen Fehlern übereinstimmen. Die Erfüllung dieser Forderung ist das zentrale Problem bei der Behandlung von Syntaxfehlern, denn je differenzierter die Methoden werden, mit deren Hilfe "Fehlertypen" diagnostiziert werden, desto schwieriger wird es, die Eigenschaften E2 und E3 zu erfüllen.

Um die Schwierigkeiten, die bei der Realisierung der Eigenschaft E1 auftreten, deutlich zu machen, geben wir einen kurzen Überblick über die bisher bekannten Lösungsmethoden. Gleichzeitig gewinnen wir dadurch die Möglichkeit, den in dieser Arbeit vorgeschlagenen Lösungsansatz vorzustellen und ihn einzuordnen in den Rahmen der bisher bekannten Methoden.

Diese können in zwei Klassen eingeteilt werden:

1. Korrekturmethoden
2. Wiederaufsetzmethoden.

Die Korrekturmethoden sind dadurch charakterisiert, daß sie die fehlerhafte Eingabezeichenfolge auf eine fehlerfreie abbilden, d.h. die Eingabezeichenfolge korrigieren. Das Problem dabei ist, die Definition der Korrekturabbildung. Dazu sind zwei grundsätzlich verschiedene Methoden bekannt:

1. Global minimale Korrektur
2. Lokale Korrektur.

Die global minimal korrigierenden Methoden, wie z.B. [Lyo,74], [Aho.Pet,72] bilden die fehlerhafte Eingabezeichenfolge auf ein Wort der Sprache ab, welches minimalen Abstand zum fehlerhaften Wort hat. Dabei versteht man unter dem Abstand die minimale Anzahl von Elementarkorrekturen die benötigt werden, um das fehlerhafte Wort auf das fehlerfreie abzubilden. Die zugelassenen Elementarkorrekturen sind: Weglassen, Ersetzen bzw. Hinzufügen eines terminalen Zeichens. Die Erfahrung zeigt, daß die Feh-

lerdiagnose, die aufgrund von global minimaler Korrektur erfolgt, im allgemeinen die tatsächlichen Fehler trifft, so daß die entsprechenden Analysealgorithmen die Eigenschaft E1 besitzen. Gleichzeitig können solche Methoden automatisch erzeugt werden. Für praktische Anwendungen sind diese Methoden nicht brauchbar, da ihr Speicherbedarf proportional n^2 und ihr Zeitbedarf proportional n^3 ist.⁽¹⁾ Der Grund dafür ist, daß zur Berechnung der global minimalen Korrekturen im allgemeinen alle Möglichkeiten der Korrektur des fehlerhaften Wortes bestimmt und gespeichert werden müssen.

Die Methoden, welche auf lokaler Korrektur beruhen, wie z.B. [LaF,71] und [Fre,64], umgehen dieses Problem dadurch, daß sie zur Entscheidung, welche Korrektur ausgeführt wird, nur die unmittelbare Umgebung der Fehlerstelle heranziehen. Die Folge davon ist, daß in vielen Fällen Korrekturen durchgeführt werden, die von den tatsächlichen Fehlern weit entfernt sind. Die entsprechenden Fehlermeldungen sind dann in der Regel irreführend oder unbrauchbar. Will man diesen Nachteil vermeiden, so muß zur Entscheidung, welche Korrektur durchgeführt wird, in Abhängigkeit von der Fehlersituation, mehr Kontext benützt werden. Dieser Sachverhalt wurde bei J.P. Lévy [Lev,71] systematisch untersucht. Dabei ergab sich, daß dieses Problem nur mit Hilfe starker Einschränkungen, nämlich für spezielle Sprachen und für eingeschränkte Fehlersituationen theoretisch lösbar ist. Für praktische Anwendungen ist die Realisierung der Lösungen jedoch zu aufwendig, insbesondere, wenn man bedenkt, daß nur für spezielle Fehlersituationen optimale Korrekturen gefunden werden.

Insgesamt heißt das, daß der Versuch der Korrektur bisher zu keiner optimalen Lösung des Fehlerbehandlungsproblems geführt hat. Der Grund dafür ist, daß die Realisierung der Korrekturabbildungen, welche mit Hilfe des Hamming-Abstandes definiert wurden, zu aufwendig ist.

Die Wiederaufsetzmethode sind dadurch gekennzeichnet, daß sie die Eingabezeichenfolge unverändert lassen. Um mehr als einen Fehler diagnostizieren zu können, wird, nachdem ein Fehler erkannt wurde, der Zustand

⁽¹⁾ n bezeichnet die Länge der Eingabezeichenfolge.

(dargestellt durch den Keller) des Analysators so verändert, daß die Eingabezeichenfolge weiter analysiert werden kann. Alle bisher veröffentlichten Wiederaufsetzverfahren beruhen auf einer oder einer Kombination der folgenden Methoden:

1. Kellergesteuertes Wiederaufsetzen
2. Zeichengesteuertes Wiederaufsetzen.

Beim kellergesteuerten Wiederaufsetzen ([Lei,70],[Rho,73]) werden die zur Fortsetzung der Analyse nötigen Kellerkorrekturen durch den aktuellen Kellerinhalt wesentlich mitbestimmt. Die beiden vielversprechendsten Strategien zur Auswahl einer Kellerkorrektur sind:

Die Verwendung einer Abstandsfunktion zur Bestimmung eines Kellerinhalts mit minimalem Abstand zum ursprünglichen.

Die Auswahl einer Kellerkorrektur mit der Eigenschaft, daß der durch sie entstehende Kellerinhalt auch durch die Analyse einer fehlerfreien Zeichenfolge entstanden sein könnte.

Beide Strategien führen zu Algorithmen mit vertretbarem Platz- und Zeitbedarf. Ihr Hauptnachteil ist, daß die von ihnen diagnostizierten Fehler von den tatsächlichen Fehlern in vielen Fällen stark abweichen. Dennoch ist das Prinzip des kellergesteuerten Wiederaufsetzens ein Fortschritt gegenüber der lokalen Korrektur, da die im Keller gespeicherte Information die Fehlersituation und Kontextsituation in komprimierter Form wiedergibt und deshalb eine effiziente Behandlung zuläßt.

Beim zeichengesteuerten Wiederaufsetzen ([Wir,68],[Rho,73],[Amm,75]) hängt die Fortsetzung der Analyse nach einem Fehler nicht mehr vom Kellerinhalt, sondern nur mehr von speziellen Zeichen in der noch zu analysierenden Zeichenfolge ab. Wird im Fehlerfall eines dieser Zeichen (genannt Wiederaufsetzzeichen) in der noch zu analysierenden Zeichenfolge gefunden, so wird der Analysator mit Hilfe von Kellerkorrekturen in einen Zustand gebracht, der auf eindeutige Weise die Fortsetzung der Analyse mit diesem Zeichen erlaubt. Das Hauptproblem dabei ist die Bestimmung der Menge von Wiederaufsetzzeichen. Die üblichen Programmiersprachen enthalten nur wenige terminale Zeichen, durch welche die Fort-

setzung der Analyse auf eindeutige Weise festgelegt ist. Das führt dazu, daß, bevor wiederaufgesetzt werden kann, große Stücke der Eingabezeichenfolge überlesen und deswegen nur wenige Fehler gefunden werden. Das heißt, die Qualität der Methode hängt davon ab, wieviele terminale Zeichen in der Sprache existieren, mit Hilfe derer eindeutig wiederaufgesetzt werden kann. Diese Eigenschaft ist unabhängig von der Grammatikklasse, so daß die Qualität der Methode von Sprache zu Sprache, bei gleicher Grammatikklasse verschieden ist. Im Hinblick auf die automatische Erzeugbarkeit von Fehlerbehandlungsalgorithmen ist dies nicht wünschenswert.

Die Diskussion der bisher bekannten Methoden kann wie folgt zusammengefaßt werden: Keine der Methoden führt zu einem Fehlerbehandlungsalgorithmus, der den Eigenschaften E1 bis E4 in befriedigender Weise gerecht wird. Insbesondere erfüllen alle Methoden, die bezüglich des Laufzeitaufwandes für Anwendungen in der Praxis geeignet sind, die Forderung E1 nur ungenügend, da sie im allgemeinen zu viele Fehler (Folgefehler) melden. Neue Lösungen des Fehlerbehandlungsproblems müssen im Bereich der Wiederaufsetzmethode gesucht werden, da insbesondere Levys Arbeit gezeigt hat, daß Korrekturmethode für praktische Anwendungen zu aufwendig sind. Die wesentlichen Nachteile der zeichengesteuerten Wiederaufsetzmethode werden dadurch verursacht, daß nicht alle terminalen Zeichen als Wiederaufsetzzeichen verwendet werden. Deshalb ist es das Ziel dieser Arbeit, für einfache Präzedenzsprachen einen Analysealgorithmus zu entwickeln und formal zu beschreiben, welcher mit jedem terminalen Zeichen wiederaufsetzt und zu prüfen, inwieweit dieser Algorithmus den Eigenschaften E1 bis E4 gerecht wird. Dabei wird die Prüfung der Eigenschaft E4 nur auf heuristische Weise geschehen können. Dies deshalb, weil für die Entscheidung der Frage, was eine wesentliche Verschlechterung der Effizienz der Analyse ist, kein allgemeingültiger Maßstab bekannt ist und die Festlegung eines solchen den Rahmen dieser Arbeit übersteigt.

Die Absicht, alle entwickelten Algorithmen formal zu beschreiben wird deshalb betont, weil in den bisher bekannten Arbeiten über Fehlerbehandlungsalgorithmen für einfache Präzedenzsprachen nur verbale Beschreibung

gen gegeben werden. Die Entwicklung von Begriffsbildungen, die eine formale Beschreibung zulassen, ist für eine saubere Darstellung von Algorithmen jedoch unerlässlich.

0.2 Abschnittsübersicht

Die Arbeit ist in sieben Abschnitte gegliedert:

In Abschnitt eins werden die in der Arbeit benötigten Begriffe definiert. Ferner wird eine Beschreibungsmethode für Analysealgorithmen für einfache Präzedenzsprachen entwickelt, die die beweistechnische Grundlage der gesamten Arbeit ist. Nach entsprechenden Modifikationen kann diese Beschreibungsmethode zur formalen Beschreibung von Bottom-Up Analysatoren im allgemeinen verwendet werden.

Abschnitt zwei beschäftigt sich mit der Erkennung von Syntaxfehlern bei einfachen Präzedenzsprachen. Es wird gezeigt, daß alle bisher bekannten Analysealgorithmen Syntaxfehler zu spät erkennen. Im Anschluß daran wird ein Analysealgorithmus entwickelt, der Syntaxfehler frühestmöglich erkennt.

Abschnitt drei beschäftigt sich mit der Behandlung von Syntaxfehlern bei einfachen Präzedenzsprachen. Zunächst wird als Grundlage für die Beurteilung von Fehlerbehandlungsalgorithmen ein Fehlerbegriff definiert. Dann werden die bisher bekannten Fehlerbehandlungsmethoden für einfache Präzedenzsprachen vorgestellt und diskutiert. Im Anschluß daran wird die mit Nachsilbenanalyse ⁽¹⁾ bezeichnete Fehlerbehandlungsmethode eingeführt sowie ihre prinzipiellen Vor- und Nachteile erörtert. Dann wird eine Klasse von Fehlerbehandlungsalgorithmen konstruiert, welche das Prinzip der Nachsilbenanalyse für einfache Präzedenzsprachen verwirklichen. Dabei wird der in Abschnitt zwei entwickelte Algorithmus, welcher Fehler frühestmöglich erkennt, eine wesentliche Rolle spielen.

In Abschnitt vier werden die in den Abschnitten zwei und drei gewonnenen Ergebnisse auf Analysatoren für einfache Präzedenzsprachen übertragen,

(1) Siehe Seite 95, Fußnote

welche Präzedenzfunktionen benützen.

In Abschnitt fünf erfolgt eine Zusammenfassung der wesentlichen Ergebnisse dieser Arbeit.

Abschnitt sechs ist das Literaturverzeichnis. Es enthält nicht nur die in dieser Arbeit zitierten Titel, sondern es wurde versucht, die wesentlichen Arbeiten zum Thema Behandlung von Syntaxfehlern vollständig zu erfassen.

Abschnitt sieben soll das Lesen dieser Arbeit erleichtern. Er enthält Verzeichnisse aller verwendeten Symbole, Mengen, Relationen, Abbildungen, Begriffe, Definitionen, Sätze, Hilfssätze, Bemerkungen, Beispiele und Bilder.

0.3 Organisatorische Übersicht

Beginnend mit Abschnitt 1.2.2 werden alle Definitionen, Sätze, Hilfssätze, Bemerkungen und Bilder mit einem Index p.k versehen. Dieser zeigt an, daß es sich um die k-te Definition (bzw. Satz, Hilfssatz usw.) des p-ten Abschnitts handelt.

Zeichenfolgen über speziellen Alphabeten werden durch kursive Schreibweise gekennzeichnet.

Verweise auf das Literaturverzeichnis haben die Form [Abc,j]. Dabei ist Abc eine Abkürzung für den oder die Verfassernamen und j das Erscheinungsjahr. Bei nur einem Verfassernamen besteht die Abkürzung Abc aus den ersten drei Buchstaben des Verfassernamens. Bei mehr als einem Verfasser besteht sie aus den ersten drei Buchstaben jedes Verfassernamens, jeweils durch Punkt getrennt (siehe Literaturverzeichnis).

Beweise werden durch die Zeichenfolge q.e.d. bzw. q.e.d. (Verweis auf abgeschlossenen Beweisteil) und Beispiele durch das Zeichen Δ abgeschlossen.

1. GRUNDLAGEN

Die in diesem Abschnitt eingeführten Grundbegriffe werden inhaltlich als bekannt vorausgesetzt. Ihre Beschreibung dient der Festlegung einer einheitlichen Notation. Soweit zweckmäßig, wurden die Begriffsbildungen, wie deren Notation, dem Werk von Aho und Ullman [Aho.Ull,72] entnommen.

1.1 Grundbegriffe aus der naiven Mengenlehre

1.1.1 Mengen

Der Begriff der Menge, wie der der Elementbeziehung sind undefinierte Grundbegriffe. Ist x ein Element der Menge M , so schreiben wir $x \in M$. Die Negation dieser Aussage wird mit $x \notin M$ beschrieben.

Die leere Menge ist diejenige Menge, welche keine Elemente hat, sie wird mit $\emptyset$ bezeichnet.

Die Menge bestehend aus den Elementen $x_1, x_2, \dots, x_n$ bezeichnen wir mit $\{x_1, \dots, x_n\}$.

Ist A eine Menge und $P(x)$ ein Prädikat für Elemente x , dann wird die Menge derjenigen Elemente mit $x \in A$ und $P(x) = \text{true}$ mit $\{x \in A \mid P(x)\}$, oder wenn klar ist, welche Menge A gemeint ist, mit $\{x \mid P(x)\}$ bezeichnet.

Die Menge A heißt eine Teilmenge der Menge B , geschrieben $A \subseteq B$, wenn jedes Element von A auch ein Element von B ist.

A heißt eine echte Teilmenge von B , geschrieben $A \subset B$, wenn $A \subseteq B$ und ein Element x existiert mit:

$$x \notin A \wedge x \in B. \quad (1)$$

Zwei Mengen A, B sind gleich, wenn gilt:

$$A \subseteq B \wedge B \subseteq A.$$

(1) $\wedge$ bedeutet: und

$\vee$ bedeutet: oder, im nicht ausschließlichen Sinn.

Sind A, B Mengen, so gilt:

1. $A \cup B := \{x \mid x \in A \vee x \in B\}$ ⁽¹⁾ ist eine Menge, sie heißt die Vereinigung von A und B,
2. $A \cap B := \{x \mid x \in A \wedge x \in B\}$ ist eine Menge, sie heißt der Durchschnitt von A und B .
3. $A \setminus B := \{x \in A \mid x \notin A \cap B\}$ heißt das Komplement von B bezüglich A.
4. A und B sind disjunkte Mengen, wenn gilt: $A \cap B = \emptyset$

Ist I eine Menge, so daß für $i \in I$, A_i eine Menge ist, so ist

$\bigcup_{i \in I} A_i := \{x \mid \exists i_0 \in I : x \in A_{i_0}\}$ ⁽²⁾ eine Menge.

Sei A eine Menge, dann ist $2^A := \{B \mid B \subseteq A\}$ eine Menge, sie heißt die Potenzmenge von A.

Seien A, B Mengen, $a \in A$ und $b \in B$, dann heißt die Menge $(a, b) := \{a, \{a, b\}\}$ das geordnete Paar von a und b.

Die Menge $A \times B := \{(a, b) \mid a \in A \text{ und } b \in B\}$ heißt das kartesische Produkt von A und B.

Die Menge der natürlichen Zahlen bezeichnen wir mit $\mathbb{N}$, $\mathbb{N} \cup \{0\}$ bezeichnen wir mit $\mathbb{N}^0$. ⁽³⁾

Die Menge der ganzen Zahlen bezeichnen wir mit $\mathbb{Z}$.

Für $n \in \mathbb{N}$ definieren wir induktiv das n-fache kartesische Produkt A^i einer Menge A:

$$A^1 := A, \quad A^{i+1} := A \times A^i \quad \text{für } i \in \mathbb{N}$$

Für A^i schreiben wir auch $\underbrace{A \times A \times A \times \dots \times A}_{i\text{-mal}}$ oder $\underbrace{AAA \dots A}_{i\text{-mal}}$.

Analog schreiben wir für $x \in A^i$ $(x_1, \dots, x_i)$ oder $x_1 \dots x_i$.

(1) := bedeutet: definiert durch

(2) $\exists$ bedeutet: es existiert
: bedeutet: mit der Eigenschaft

(3) Die Zeichen 0 und O werden zur Bezeichnung der Ziffer null verwendet. o in der Regel als Index. Die Buchstaben h, i, j, k, m, n, p, q, r, s, t bezeichnen, wenn nicht explizit anders vereinbart, Elemente aus $\mathbb{N}^0$.

Ist $x \in A^i$ und $x = (x_1, \dots, x_i) = x_1 \dots x_i$, dann ist das Spiegelbild x^s von x definiert durch $x^s := (x_i, \dots, x_1)$

Selbstverständlich gelten die oben definierten Schreibweisen auch für kartesische Produkte von endlich vielen verschiedenen Mengen.

1.1.2 Relationen

Seien A, B Mengen.

Eine Relation von A nach B ist eine Teilmenge von $A \times B$.

Eine Relation auf A ist eine Relation von A nach A .

Ist R eine Relation, so schreiben wir auch $a R b$ oder $a \in R(b)$ für $(a, b) \in R$.

Sei R eine Relation auf A dann gilt:

1. R heißt reflexiv : $\forall^{(1)} a R a \quad \forall a \in A$
2. R heißt symmetrisch : $a R b \} b R a \quad \forall a, b \in A$
3. R heißt transitiv : $a R b \wedge b R c \} a R c \quad \forall a, b, c \in A$

Sei R eine Relation auf A , und $k \in \mathbb{N}$, dann ist das k -fache Produkt R^k von R wie folgt rekursiv definiert:

1. $a R^1 b$: $a R b$
2. Für $i \in \mathbb{N}$ gilt $a R^i b$: $\exists c \in A : a R c \wedge c R^{i-1} b$

Sei R eine Relation auf A , dann heißt:

1. R^+ := $\bigcup_{i \in \mathbb{N}} R^i$ die transitive Hülle von R
2. R^* := $R^+ \cup R^0$, wo R^0 := $\{(a, a) \mid a \in A\}$ die transitive und reflexive Hülle von R .

¹ $\forall$ bedeutet: per definitionem
 $\}$ bedeutet: dann und nur dann, wenn gilt.
 $\}$ bedeutet: es folgt
 $\forall$ bedeutet: für alle

Bem.: R^+ (bzw. R^*) sind die kleinsten transitiven (bzw. transitiven und reflexiven) Relationen, die R enthalten.

1.1.3 Abbildungen

Eine Abbildung (oder Funktion) F einer Menge A nach einer Menge B ist eine Relation von A nach B mit der Eigenschaft: $(a,b) \in F \wedge (a,c) \in F \} b=c$.

Ist $(a,b) \in F$, dann schreiben wir auch $F(a) = b$.

Ist $F(a)$ für alle $a \in A$ definiert, so ist F eine totale, andernfalls eine partielle Abbildung von A nach B . In beiden Fällen schreiben wir $F: A \rightarrow B$, A heißt der Urbildbereich, B der Bildbereich von F .

Eine Abbildung $F: A \rightarrow B$ heißt

1. injektiv : $\times$ zu $b \in B$ existiert höchstens ein $a \in A$ mit:
 $F(a) = b$
2. surjektiv: $\times$ zu $b \in B$ existiert ein $a \in A$ mit: $F(a) = b$
3. bijektiv : $\times$ F ist total, injektiv, surjektiv.

Mit Hilfe des Begriffs der bijektiven Abbildung wird der Begriff der Mächtigkeit von Mengen definiert.

Zwei Mengen A und B heißen gleich mächtig

: $\times$ Es existiert eine bijektive Abbildung $M: A \rightarrow B$

Eine Menge M ist endlich, wenn sie für ein $n \in \mathbb{N}$ gleichmächtig zur Menge $\{1, \dots, n\}$ ist.

Unter der Mächtigkeit einer endlichen Menge verstehen wir die Anzahl ihrer Elemente und bezeichnen diese mit $|M|$.

Eine Menge M heißt abzählbar, wenn sie gleichmächtig zur Menge der natürlichen Zahlen ist.

Eine Menge heißt überabzählbar, wenn sie weder endlich noch abzählbar ist.

Sei A eine Menge, I eine unendliche, J ($|J| = n$) eine endliche Teilmenge der ganzen Zahlen $\mathbb{Z}$, dann heißt jede Abbildung $f: I \rightarrow A$ eine unendliche Folge über A und jede Abbildung $f: J \rightarrow A$ eine endliche Folge über A . In

beiden Fällen schreiben wir für $f(n) = a$ auch $\underline{a_n}$.

Das Spiegelbild f^s der endlichen Folge $f : I \rightarrow A$ mit $I = \{1, \dots, n\}$ ist definiert durch $f^s(j) := f(n-j+1)$ für $j = 1, \dots, n$.

Sei A eine endliche Menge, dann verstehen wir unter einer Numerierung Num_A von A jede bijektive Abbildung $\text{Num}_A : A \rightarrow \{1, 2, \dots, |A|\}$.

Ist klar um welche Menge A es sich handelt, so lassen wir den Index A weg. Die Mengen $\{1, 2, \dots, k\}$ bzw. $\{0, 1, \dots, k\}$ bezeichnen wir mit $\mathbb{N}_k$ bzw. $\mathbb{N}_k^0$.

1.2 Grundbegriffe aus der Theorie der Sprachübersetzung

1.2.1 Formale Sprachen

Sei A eine Menge.

Die Menge aller nichtleeren Worte über A , bezeichnet mit A^+ , ist definiert durch:

$$A^+ := \bigcup_{i \in \mathbb{N}} A^i.$$

Ist $x \in A^+$, so existiert genau ein n mit $x \in A^n$. Die Zahl n heißt dann die Länge von x und wird mit $|x|$ bezeichnet.

Für $x, y \in A^+$ mit $x = x_1 \dots x_n$, $y = y_1 \dots y_m$, $n, m \in \mathbb{N}$ definieren wir die Konkatenation $x \circ y$ von x mit y durch:

$$x \circ y := x_1 \dots x_n y_1 \dots y_m$$

Statt $x \circ y$ schreiben wir i.allg. xy .

Die Konkatenation $\circ$ ist eine Abbildung von $A^+ \times A^+$ nach A^+ .

Das Paar $(A^+, \circ)$ heißt die von A erzeugte freie Halbgruppe ohne Einselement.

Erweitert man diese um ein Einselement ε , durch

$$A^* := A^+ \cup \{\varepsilon\} \quad (\text{wo } A^+ \cap \{\varepsilon\} = \emptyset) \quad \text{und}$$

$$x \circ \varepsilon := \varepsilon \circ x := x \quad \forall x \in A^+,$$

so erhält man mit $(A^*, \circ)$ die von A erzeugte freie Halbgruppe.

Im weiteren werden mit A^+ bzw. A^* immer die Paare $(A^+, \circ)$ bzw. $(A^*, \circ)$ bezeichnet.

A^* heißt auch die Menge aller (endlichen) Zeichenfolgen (Wörter) über A .

Das Einselement ϵ heißt die leere Zeichenfolge und hat die Länge null.

Unter einem Alphabet verstehen wir eine endliche Menge.

Eine formale Sprache $S^{(1)}$ über dem Alphabet A ist eine Teilmenge von A^* .

Sei Γ ein Alphabet, $\alpha, \beta \in \Gamma^*$, $X \in \Gamma$, dann gilt:

1. α heißt Präfix von β , wenn ein $\gamma \in \Gamma^*$ existiert mit $\alpha\gamma = \beta$.
Die Menge aller Präfixe von β bezeichnen wir mit Präfix (β).
2. α heißt Infix von β , wenn $\gamma, \delta \in \Gamma^*$ existieren mit $\gamma\alpha\delta = \beta$.
Die Menge aller Infixe von β bezeichnen wir mit Infix (β).
3. α heißt Suffix von β , wenn ein $\gamma \in \Gamma^*$ existiert mit $\gamma\alpha = \beta$.
Die Menge aller Suffixe von β bezeichnen wir mit Suffix (β).
4. X heißt Anfang von β (i.Z. $X = \text{Anf}(\beta)$), wenn ein $\gamma \in \Gamma^*$ existiert mit $X\gamma = \beta$.
5. X heißt Ende von β (i.Z. $X = \text{End}(\beta)$), wenn ein $\gamma \in \Gamma^*$ existiert mit $\gamma X = \beta$.

Sei Σ ein Alphabet, L Sprache über Σ und $u \in \Sigma^*$, dann gilt:

1. u heißt Präfix von L , wenn ein $v \in \Sigma^*$ existiert mit $uv \in L$.
Die Menge aller Präfixe von L bezeichnen wir mit Präfix (L).
2. u heißt Infix von L , wenn $v, w \in \Sigma^*$ existieren mit $vuw \in L$.
Die Menge aller Infixe von L bezeichnen wir mit Infix (L).
3. u heißt Suffix von L , wenn ein $v \in \Sigma^*$ existiert mit $vu \in L$.
Die Menge aller Suffixe von L bezeichnen wir mit Suffix (L).

1.2.2 Sprachdefinition mit Hilfe von kontextfreien Sprachen

Die wichtigste Methode zur Definition der Syntax von Programmiersprachen beruht auf dem Begriff der kontextfreien Grammatik.

⁽¹⁾ Im folgenden bezeichnen wir formale Sprachen oft einfach als Sprachen.

Definition 1.1

Unter einer kontextfreien Grammatik (KFG) verstehen wir ein Quadrupel $G = (N, \Sigma, P, S)$ mit den Eigenschaften:

1. N ist das Alphabet der nichtterminalen Zeichen (oder syntaktischen Variablen)
2. Σ ist das Alphabet der terminalen Zeichen, wobei N und Σ disjunkt sind.
3. P ist eine endliche Teilmenge von $N \times \Gamma_G^*$, wo $\Gamma_G := N \cup \Sigma$ gilt. Ein Element (A, β) von P heißt eine Produktion und wird auch mit $A \rightarrow \beta$ bezeichnet. Γ_G heißt das Alphabet der Grammatik G .⁽¹⁾
4. S ist ein Element von N und heißt das Startsymbol.

Definition 1.2

Sei $G = (N, \Sigma, P, S)$ eine KFG, dann definieren wir die Relationen $\xRightarrow{G}$, $\xRightarrow{R, G}$ auf Γ^* durch:

1. $\delta \xRightarrow{G} \eta : \nexists \alpha, \beta, \gamma \in \Gamma^* \wedge A \in N : \delta = \alpha A \gamma \wedge \eta = \alpha \beta \gamma \wedge A \rightarrow \beta \in P$
2. $\delta \xRightarrow{R, G} \eta : \nexists \alpha, \beta \in \Gamma^* \wedge u \in \Sigma^* \wedge A \in N : \delta = \alpha A u \wedge \eta = \alpha \beta u \wedge A \rightarrow \beta \in P$

Die transitive Hülle von $\xRightarrow{G}$, bzw. $\xRightarrow{R, G}$ bezeichnen wir mit $\xRightarrow{G}^+$, bzw. $\xRightarrow{R, G}^+$.

Die transitive und reflexive Hülle von $\xRightarrow{G}$, bzw. $\xRightarrow{R, G}$ bezeichnen wir mit $\xRightarrow{G}^*$, bzw. $\xRightarrow{R, G}^*$.

Für $\delta \xRightarrow{G} \eta$, bzw. $\delta \xRightarrow{R, G} \eta$, bzw. $\delta \xRightarrow{G}^* \eta$ führen wir die Sprechweise δ produziert direkt η oder η ist direkt ableitbar aus δ bzw. δ produziert

(1) Für Γ_G schreiben wir Γ , wenn eindeutig ist, zu welcher Grammatik G Γ Alphabet ist. Analoges gilt für alle Bezeichnungen, wo Indizierungen dieser Art auftreten.

nichttrivial η oder η ist nichttrivial ableitbar aus δ bzw. δ produziert η oder η ist ableitbar aus δ , ein.

Gilt $\delta \xrightarrow{G} \eta$ und $\delta = \alpha A \gamma$, $\eta = \alpha \beta \gamma$, $A \rightarrow \beta \in P$, dann sagen wir, δ produziert η durch Ableitung der Variablen A (auf β) oder η wird reduziert auf δ durch Reduktion von β (auf A).

Unter der von einer Grammatik erzeugten Sprache verstehen wir die Menge der terminalen Zeichenfolgen, die aus der Startvariablen ableitbar sind.

Definition 1.3

Sei $G = (N, \Sigma, P, S)$ eine KFG.

Die Menge $L(G)$ definiert durch

$$L(G) := \{u \in \Sigma^* \mid \exists S \xrightarrow{G}^* u\}$$

heißt die von G erzeugte Sprache.

Jedes $u \in L(G)$ heißt ein Satz bzw. ein Wort von $L(G)$.

Um festzustellen, ob für eine Zeichenfolge $u \in \Sigma^*$, $S \xrightarrow{G}^* u$ gilt, bildet man Ableitungen von u aus S .

Definition 1.4

Sei $G = (N, \Sigma, P, S)$ eine KFG, $\alpha, \beta \in \Gamma^*$, dann gilt:

1. Unter einer Ableitung (der Länge $k \in \mathbb{N}$ (bzgl. G)) ⁽¹⁾ von β aus α verstehen wir jede Folge $\omega_i \in \Gamma^*$, $i = 0, \dots, k$ mit den Eigenschaften:

$$1.1 \quad \omega_0 = \alpha$$

$$1.2 \quad \omega_i \xrightarrow{G} \omega_{i+1} \quad \text{für } i = 0, \dots, k-1$$

$$1.3 \quad \omega_k = \beta$$

(1) Teile von Definitionen, welche geklammert sind, sind optional.

2. Unter einer Rechtsableitung von β aus α verstehen wir eine Ableitung ω_i , $i = 0, \dots, k$ von β aus α , wo $\omega_i \xrightarrow[\text{G}]{\text{R}} \omega_{i+1}$ gilt.
3. Unter einer Ableitung (der Länge $k \in \mathbb{N}$ bzgl. G) von α verstehen wir eine Ableitung (der Länge k bzgl. G) von α aus S. Hat $\alpha \in \Gamma^*$ eine Ableitung aus S dann heißt α eine Satzform (von G).
4. Unter einer Rechtsableitung von $\alpha \in \Gamma^*$ verstehen wir eine Rechtsableitung von α aus S. $\alpha \in \Gamma^*$ heißt eine rechtskanonische Satzform oder Rechtssatzform, wenn α eine Rechtsableitung besitzt.

Die folgende Definition ist von grundlegender Bedeutung, für die in dieser Arbeit verwendeten Beweistechniken (siehe Abschnitt 1.3). Durch sie wird festgelegt, was wir unter einem Analysator bzw. einem Analysealgorithmus verstehen wollen.

Definition 1.5

Sei L formale Sprache über dem Alphabet Σ . FM eine Menge mit $FM \cap \{\text{true}\} = \emptyset$.

Eine Abbildung $A : \Sigma^* \rightarrow \{\text{true}\} \cup FM^+$ heißt genau dann ein Analysator, bzw. ein Analysealgorithmus für L, wenn gilt:

$$A(u) = \begin{cases} \text{true} & \text{falls } x \in L \\ f \in FM & \text{falls } x \notin L \end{cases}$$

FM heißt die Menge der Fehlermeldungen von A.

Bemerkung 1.1

1. Üblicherweise bestimmen Analysatoren von Sprachen, die von Grammatiken erzeugt sind, auch den "Strukturbaum" des zu analysierenden Wortes. Darauf wurde verzichtet, da die Strukturbäume im Rahmen dieser Arbeit bedeutungslos sind.

2. In der Praxis werden Analysatoren durch Algorithmen realisiert. Wir bezeichnen solche Algorithmen ebenfalls als Analysatoren.
3. Für die Menge FM werden wir in dieser Arbeit stets die Menge der natürlichen Zahlen wählen. Dabei sollen für $x \in \Sigma^+ \setminus L$ mit $x = a_1 \dots a_n$ die folgenden Konventionen gelten:

$$3.1 \quad A(x) \leq n+1$$

3.2 $A(x) = k \leq n$ soll bedeuten, daß a_k dasjenige Zeichen ist, bei dessen Analyse der Fehler erkannt wurde

3.3 $A(x) = n+1$ soll bedeuten, daß in $a_1 \dots a_n$ erst dann ein Fehler entdeckt wurde, nachdem a_n als das letzte Zeichen von $a_1 \dots a_n$ erkannt wurde.

Eine andere Möglichkeit der Wahl von FM ist $FM = \{\text{false}\}$. Sie wird oft in theoretischen Arbeiten benutzt, ist aber für praktische Zwecke unbrauchbar, weil sie über den Ort der Fehlererkennung keine Aussagen zuläßt.

Bezüglich der Notation von Produktionen bzw. von Grammatiken treffen wir die folgenden Übereinkünfte:

1. Existieren in einer Grammatik n Produktionen der Gestalt

$$\begin{array}{c} A \rightarrow \beta_1 \\ \vdots \\ A \rightarrow \beta_n \end{array}, \quad \beta_i \in \Gamma^*$$

so dürfen diese auch in der Form

$$A \rightarrow \beta_1 \mid \beta_2 \mid \dots \mid \beta_n$$

notiert werden.

2. Wenn nicht explizit anders vereinbart, so repräsentieren bei geg. Grammatik $G = (N, E, P, S)$

2.1 a, b, c und d Elemente aus E

2.2 A, B, C, D und S Elemente aus N

2.3 $U, V, \dots Z$ Elemente aus Γ

2.4 $\alpha, \beta, \dots$ Elemente aus Γ^*

2.5 $u, v, \dots$ Elemente aus Σ^*

2.6 Zwischenräume sind Trennzeichen und sonst bedeutungslos.

3. Treten bei der Beschreibung von Produktionen die Zeichen $\rightarrow$ oder $|$ als terminale Zeichen auf, so werden diese überstrichen.
4. Grammatiken werden i.allg. nur durch die Angabe der Produktionen definiert, wobei eine der Produktionen mit der Startvariablen auf der linken Seite als erste notiert wird.
5. Indizierungen ändern nichts an diesen Übereinkünften.

Definition 1.6

Sei $G = (N, \Sigma, P, S)$ eine KFG, $A \rightarrow \beta \in P$, $X \in \Gamma$ dann gilt:

1. Die Variable $LS(A \rightarrow \beta) := A$ heißt die linke Seite von $A \rightarrow \beta$.
2. Die Zeichenfolge $RS(A \rightarrow \beta) := \beta$ heißt die rechte Seite von $A \rightarrow \beta$.
3. Die Menge Prefix $(A \rightarrow \beta) := \{\alpha \in \Gamma^* \mid \alpha \in \text{Prefix}(\beta)\}$ heißt die Menge der Präfixe von $A \rightarrow \beta$.
4. Die Menge Suffix $(A \rightarrow \beta) := \{\alpha \in \Gamma^* \mid \alpha \in \text{Suffix}(\beta)\}$ heißt die Menge der Suffixe von $A \rightarrow \beta$.
5. Die Menge Infix $(A \rightarrow \beta) := \{\alpha \in \Gamma^* \mid \alpha \in \text{Infix}(\beta)\}$ heißt die Menge der Menge der Infixe von $A \rightarrow \beta$.
6. Sei $X \in \Gamma$ und $A \in N$, dann definiert man die Relation Anf, End über Γ wie folgt:

$$\begin{aligned} X \text{ Anf } A &: \text{ } \nexists \text{ } A \xrightarrow[G]{*} X\beta, \text{ wo } \beta \in \Gamma^* \\ X \text{ End } A &: \text{ } \nexists \text{ } A \xrightarrow[G]{*} \beta X, \text{ wo } \beta \in \Gamma^* \end{aligned}$$

Beispiel 1.1: Die Sprache L1

Die im folgenden definierte Sprache L1 wird innerhalb dieser Arbeit zur Formulierung von Beispielen benützt. Sie ist so gewählt, daß die wesentlichen Typen von Syntaxfehlern, die in Programmen, formuliert in üblichen Programmiersprachen, auftreten, auch mit Hilfe von L1-Programmen konstruiert werden können. Ausgenommen davon sind alle Fehler, welche üblicherweise vom Vorgruppierer⁽¹⁾ behandelt werden. L1 wird mit Hilfe von zwei äquivalenten Grammatiken G0 und G1 definiert. Diese sind in den Bildern 1.1 und 1.2 dargestellt.

An Hand der Grammatik G0 erkennt man die Struktur der Sprache L1. Sie ist eine Teilmenge der Sprache PASCAL ([Wir.Jen,74]), wobei für Namen, bzw. Typen nur die Zeichenfolge `id` zugelassen ist. Eine L1-Zeichenfolge besteht aus dem Programmkopf, dem eine optionale Variablendeklaration, eine Block-Anweisung und ein Punkt folgen. Eine Block-Anweisung ist eine Folge von Anweisungen, jeweils durch Strichpunkt getrennt. Die möglichen Anweisungen sind: Zuweisungs-Anweisung, Block-Anweisung, If-Anweisung, While-Anweisung. Bei der syntaktischen Beschreibung der Ausdrücke wurde bewußt auf die Festlegung von Operatorvorrangregeln verzichtet, da diese im Rahmen der Behandlung von Syntaxfehlern bedeutungslos sind.

Beispiel 1.3 enthält eine einfache Zeichenfolge der Sprache L1.

Die Grammatik G1 ist eine einfache Präzedenzgrammatik für L1. Alle Aussagen über L1-Zeichenfolgen beziehen sich auf die Grammatik G1.

(1) Unter dem Vorgruppierer versteht man denjenigen Teil eines Übersetzers, welcher die Zeichen der Eingabezeichenfolge zu den syntaktischen Grundeinheiten der Sprache, wie z.B. Namen, Schlüsselwörter, Zahlen usw. zusammenfaßt (siehe [See,74]).

Bild 1.1: Die Grammatik GO

GO ist durch die folgenden Produktionen definiert.

$$\begin{aligned} S &\rightarrow \text{program id ; } B . \\ B &\rightarrow V C \mid C \\ V &\rightarrow \text{var } Vl : id ; \\ Vl &\rightarrow id \mid id , Vl \\ C &\rightarrow \text{begin } L \text{ end} \\ L &\rightarrow St \mid St ; L \\ St &\rightarrow C \mid A \mid I \mid W \\ A &\rightarrow id := E \mid id[E] := E \\ I &\rightarrow \text{if } E \text{ then } St \mid \text{if } E \text{ then } St \text{ else } St \\ W &\rightarrow \text{while } E \text{ do } St \\ E &\rightarrow T = T \mid T \\ T &\rightarrow F \mid T + F \mid T * F \\ F &\rightarrow id \mid (E) \mid id[E] \end{aligned}$$

Bild 1.2: Die Grammatik G1 der Sprache L1

G1 ist definiert durch die folgenden Produktionen, wobei die in dieser Arbeit verwendete Numerierung der Produktionen von G1 links angegeben ist.

- 1 $S \rightarrow Ph\ B$
- 2 $Ph \rightarrow Pi;$
- 3 $Pi \rightarrow program\ id$
- 4 $B \rightarrow V\ Ch \mid Ch$
- 5 $V \rightarrow Vh;$
- 6 $Vh \rightarrow var\ Il1\ id$
- 7 $Il1 \rightarrow Il$
- 8 $Il \rightarrow id : \mid id, Il$
- 9 $Ch \rightarrow C.$
- 10 $C \rightarrow begin\ L1\ end$
- 11 $L1 \rightarrow L$
- 12 $L \rightarrow L2\ St \mid St$
- 13 $L2 \rightarrow L;$
- 14 $St \rightarrow Os \mid Cs$
- 15 $Os \rightarrow Ic\ else\ Os \mid Ic \mid I\ Os \mid W\ do\ Os$
- 16 $Cs \rightarrow Ic\ else\ Cs \mid W\ do\ Cs \mid A \mid C$
- 17 $Ic \rightarrow I\ Cs$
- 18 $I \rightarrow if\ E\ then$
- 19 $W \rightarrow while\ E$
- 20 $A \rightarrow id : G\ E \mid id[E] : G\ E$
- 21 $G \rightarrow =$
- 22 $E \rightarrow Th\ T1 \mid T1$
- 23 $T1 \rightarrow T$
- 24 $Th \rightarrow T =$
- 25 $T \rightarrow F \mid T + F \mid T * F$
- 26 $F \rightarrow id \mid (E) \mid id\ [E]$

d.h. $G1 = (N1, \Sigma1, P1, S)$ mit:

- $N1 = \{ S, Ph, B, Pi, V, Vh, Il1, Il, Ch, C, L1, L, L2, St, Os, Cs, Ic, I, W, A, G, E, T1, Th, T, F \}$
- $\Sigma1 = \{ ;, , program, id, var, :, ,, ., begin, end, else, do, if, then, while, [,], =, +, *, (,) \}$

1.2.3 Einfache Präzedenzsprachen

Analysatoren, welche allgemeine kontextfreie Sprachen analysieren, sind wegen ihres Platz- und Zeitbedarfes für praktische Anwendungen nicht geeignet. Um weniger aufwendigere Analysatoren zu erhalten, wurden kontextfreie Grammatiken auf die verschiedensten Weisen eingeschränkt. Einen besonders einfachen und effizienten Analysealgorithmus erlauben die von Wirth und Weber [Wir.Web,66] definierten einfachen Präzedenzsprachen.

Definition 1.7

Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik, $X, Y \in \Gamma$, $a \in \Sigma$. Dann definiert man die (Wirth-Weber) Präzedenzrelationen $\triangleleft, \triangle, \triangleright$ auf $\Gamma = \Sigma \cup N$ wie folgt:

1. $X \triangleleft Y : \begin{matrix} \text{X} \\ \text{Y} \end{matrix} \quad \exists A \rightarrow \alpha X B \beta \text{ in } P: \exists B \xrightarrow{+} Y \gamma$
2. $X \triangle Y : \begin{matrix} \text{X} \\ \text{Y} \end{matrix} \quad \exists A \rightarrow \alpha X Y \beta \text{ in } P$
3. $X \triangleright a : \begin{matrix} \text{X} \\ \text{Y} \end{matrix} \quad \exists A \rightarrow \alpha B C \beta \text{ in } P: \exists B \xrightarrow{+} \gamma X \wedge C \xrightarrow{*} a \delta$
, wobei $A, B, C \in N$, $\alpha, \beta, \gamma, \delta \in \Gamma^*$ gilt.

Bemerkung 1.2

1. Die oben gegebene Definition ist äquivalent zu der von Wirth-Weber vorgeschlagenen.
2. Besteht zwischen zwei Zeichen $X, Y \in \Gamma$ einer kontextfreien Grammatik keine Präzedenzrelation, so notieren wir das in der Form $X ? Y$.

Definition 1.8

Eine Präzedenzgrammatik ist eine kontextfreie Grammatik $G = (N, \Sigma, P, S)$ mit folgenden Eigenschaften:

1. G enthält keine ϵ -Produktionen⁽¹⁾ und keine Produktionen der Gestalt $A \rightarrow \alpha S \beta$ mit $\alpha, \beta \in (\Gamma - \{S\})^*$
2. G ist reduziert, d.h. jede Variable tritt in mindestens einer Satzform auf und aus jeder Variablen kann mindestens eine nichtleere terminale Zeichenfolge abgeleitet werden (d.h. zu $A \in N \exists S \xRightarrow{*} \alpha A \beta \wedge A \xRightarrow{*} u$).

⁽¹⁾ D.h. keine Produktionen der Gestalt $A \rightarrow \epsilon$.

3. Zu keiner Variablen A gibt es eine Ableitung $A \Rightarrow^+ A$.
4. Zwischen zwei Zeichen $X, Y \in \Gamma$ gilt höchstens eine Wirth-Weber Präzedenzrelation.

Definition 1.9

Eine Präzedenzgrammatik G heißt einfache Präzedenzgrammatik (EPG), wenn sie eindeutig umkehrbar ist, d.h. für beliebige Produktionen

$A \rightarrow \alpha, B \rightarrow \beta$ gilt: $\alpha = \beta \} A = B$.

Definition 1.10

Eine Sprache S heißt eine einfache Präzedenzsprache (EPS), wenn eine EPG G existiert mit: $L(G) = S$.

Beispiel 1.2

Die Grammatik G1 aus Beispiel 1.1 ist eine EPG, d.h. L1 ist eine EPS. Üblicherweise werden die Präzedenzrelationen einer EPG mit Hilfe einer Matrix, der Relationenmatrix, dargestellt. Die Matrix für G1 ist in Bild 1.3 dargestellt.

Δ

Auffällig in Bild 1.3 ist die Spalte (bzw. Zeile) für das neu hinzugefügte Randzeichen #. Dieses Randzeichen # wird von den in der Praxis verwendeten Analysatoren zum Erkennen von Anfang und Ende der Eingabezeichenfolge benötigt. Das Randzeichen ist disjunkt zum Gesamtalphabet gewählt, und genügt per definitionem den Relationen:

$$\begin{array}{ll} 1. \# \prec X : \frac{Y}{X} & \exists S \Rightarrow^+ X\alpha \\ 2. Y \succ \# : \frac{Y}{X} & \exists S \Rightarrow^+ \alpha Y \end{array}$$

Es wird im folgenden stets angenommen, daß jede Zeichenfolge, welche mit Hilfe eines Analysators analysiert wird, links und rechts von je einem Randzeichen begrenzt wird.

Ist $G = (N, \Sigma, P, S)$ eine EPG, dann gelten die folgenden Schreibweisen:

1. $\# \Sigma^* := \{ \# u \mid u \in \Sigma^* \}$
2. $\Sigma^* \# := \{ u \# \mid u \in \Sigma^* \}$
3. $\# \Sigma^* \# := \{ \# u \# \mid u \in \Sigma^* \}$
4. $\# \Gamma^*, \Gamma^* \#, \# \Gamma^* \#$ analog

Ferner erweitern wir die Relationen $\Rightarrow$, $\xRightarrow{R}$ über Γ^* wie folgt zu Relationen über $\# \Gamma^* \#$. Seien $\alpha, \beta \in \Gamma^*$ dann gilt:

1. $\# \alpha \# \rightarrow \# \beta \# : \bigvee \exists \alpha \rightarrow \beta$
2. $\# \alpha \# \xrightarrow{R} \# \beta \# : \bigvee \exists \alpha \xrightarrow{R} \beta$

Bild 1.3: Matrix der Präzedenzrelationen für L1 bzgl. G1 ⁽¹⁾

A B C C C E F G I I I I L L L O P P S S T T T V V W b d e e n d i f p # t v w : + * () = , . [] ;
h s c l l l 1 2 s h i t h l h e o e n d f t h e r i l e g i n r a m

[illegible]

(1) Die Relationen $\dot{=}$, $\dot{<}$, $\dot{>}$ wurden aus Gründen der Lesbarkeit in der Form $=$, $<$, $>$ notiert.

1.3 Die Beschreibung von Analysealgorithmen für einfache Präzedenzsprachen

Ziel dieser Arbeit ist die Entwicklung von "Analysealgorithmen". Die Beschreibung von solchen Algorithmen ist problematisch. Wählt man zur Darstellung einen geeigneten Dialekt einer höheren Programmiersprache, so sind formale Beweise über deren Eigenschaften, wenn überhaupt, nur mehr mit einem Aufwand möglich, der den Rahmen dieser Arbeit sprengt ("Program Proving"). Deshalb können in diesem Fall Beweise nur heuristisch geführt werden. Aus diesem Grund werden wir in dieser Arbeit, wie bereits durch Definition 1.5 angedeutet, Analysealgorithmen in der Form von Abbildungen darstellen. Alle Behauptungen können dann exakt bewiesen werden. Der Nachteil dieser Beschreibungsmethode ist, daß konkrete Hinweise auf Realisierungen der Algorithmen in der Form von Programmen, sowie auf deren dynamische Eigenschaften verloren gehen. Für die in dieser Arbeit auftretenden Analysatoren ist dieser Nachteil nicht gravierend, da ihre Implementierungen keine prinzipiellen Probleme mit sich bringen und aus der Darstellung der Abbildungen der Ablauf der entsprechenden Programme erkennbar wird.

Als Beispiel für das oben Gesagte wollen wir in den drei folgenden Abschnitten eine Modifikation des für diese Arbeit grundlegenden Analysators von Wirth und Weber [Wir.Web,66] für einfache Präzedenzsprachen auf drei verschiedene Weisen darstellen: verbal, in der Form eines symbolischen Programms, mit Hilfe einer Abbildung.

Die dabei in Abschnitt 1.3.3 verwendete Methode der Beschreibung von Analysatoren mit Hilfe von Abbildungen ist verwandt mit der von Aho und Ullman [Aho,Ull,72] entwickelten Methode. Sie ist die notationelle Basis für die Darstellung sämtlicher in dieser Arbeit behandelter bzw. entworfener Analysatoren.

1.3.1 Anschauliche Beschreibung

Der Syntaxanalysealgorithmus für einfache Präzedenzsprachen ist ein "Bottom-Up" Verfahren. Solche Verfahren versuchen zur gegebenen Eingabe-

zeichenfolge eine Rechtsanalyse zu bilden. Unter einer Rechtsanalyse versteht man eine Folge von Produktionen, die von rechts nach links gelesen, eine Rechtsableitung darstellen. Ist also π eine Rechtsableitung, so ist das Spiegelbild π^s eine Rechtsanalyse.

Beispiel 1.3: Rechtsanalyse zu gegebener Ll-Zeichenfolge

```
program id ;
begin
    id := id + id
end
```

Rechtsableitung:

durch Anwendung von Produktion i

$S \rightarrow Ph \ B$	1
$\rightarrow Ph \ Ch$	4
$\rightarrow Ph \ C \ .$	9
$\rightarrow Ph \ begin \ L1 \ end.$	10
$\rightarrow Ph \ begin \ L \ end.$	11
$\rightarrow Ph \ begin \ St \ end.$	12
$\rightarrow Ph \ begin \ Cs \ end.$	14
$\rightarrow Ph \ begin \ A \ end.$	16
$\rightarrow Ph \ begin \ id : G \ E \ end.$	20
$\rightarrow Ph \ begin \ id : G \ T1 \ end.$	22
$\rightarrow Ph \ begin \ id : G \ T \ end.$	23
$\rightarrow Ph \ begin \ id : G \ T + F \ end.$	25
$\rightarrow Ph \ begin \ id : G \ T + id \ end.$	26
$\rightarrow Ph \ begin \ id : G \ F + id \ end.$	25
$\rightarrow Ph \ begin \ id : G \ id + id \ end.$	26
$\rightarrow Ph \ begin \ id := id + id \ end.$	21
$\rightarrow Pi ; \ begin \ id := id + id \ end.$	2
$\rightarrow program \ id ; \ begin \ id := id + id \ end.$	3

Die Rechtsableitung ist also durch

$\pi = 1, 4, 9, 10, 11, 12, 14, 16, 20, 22, 23, 25, 26, 25, 26, 21, 2, 3$ gegeben.

Die Rechtsanalyse ist dann durch

$\pi^s = 3, 2, 21, 26, 25, 26, 25, 23, 22, 20, 16, 14, 12, 11, 10, 9, 4, 1$ gegeben.

Δ

Grundlegend für die Konstruktion einer Rechtsanalyse ist der Begriff der Primphrase einer Rechtssatzform. Er ist wie folgt definiert:

Definition 1.11

Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik und gelte

$$\exists S \xrightarrow{R}^* \alpha A w \xrightarrow{R} \alpha \beta w \quad \text{mit } \alpha, \beta \in \Gamma^*, w \in \Sigma^*, A \rightarrow \beta \in P$$

dann heißt β eine Primphrase der rechtskanonischen Satzform $\alpha \beta w$.

Das heißt, unter einer Primphrase einer Rechtssatzform versteht man jedes ihrer Infixe, welches rechte Seite einer Produktion ist, und durch die linke Seite dieser Produktion ersetzt werden kann, so daß die sich dabei ergebende Zeichenfolge eine Rechtssatzform ist.

Aus der Definition des Begriffes Primphrase folgt unmittelbar, daß eine Rechtsanalyse eines Wortes einer kontextfreien Sprache dadurch gewonnen werden kann, daß ausgehend von dem Wort selbst, sukzessive Primphrasen reduziert werden, bis die Startvariable der Grammatik gewonnen ist.

Beispiel 1.4: Reduktion der Primphrase

$\underbrace{\text{program } id;}_{P_i} \text{ begin } id := id + id \text{ end.}$	Primphrase <i>program id</i> wird reduziert auf P_i
$\underbrace{P_i}_{P_h} ; \text{ begin } id := id + id \text{ end.}$	Primphrase P_i ; wird reduziert auf P_h
$P_h \quad \text{begin } id := id + id \text{ end.}$	Primphrase = wird reduziert auf G
$P_h \quad \text{begin } id : G \text{ id} + id \text{ end.}$	
$\vdots$	
S	Δ

Die im Abschnitt 1.2.3 definierten Präzedenzrelationen haben folgende Bedeutung ([Gri,71]).

1. $X \triangleleft Y$: Y Anfang einer Primphrase von G
2. $X \trianglelefteq Y$: Die Zeichenfolge XY ist Teilzeichenfolge einer Primphrase von G
3. $X \triangleright a$: X ist Ende einer Primphrase von G .

Deswegen ergibt sich unter Verwendung von Präzedenzrelationen für eine Eingabezeichenfolge $\# a_1 \dots a_n \#$ folgender Analysealgorithmus:

1. Suche, beginnend mit der Eingabezeichenfolge, in der jeweils aktuellen Satzform $X_0 X_1 \dots X_n X_{n+1}$, $X_i \in \Gamma$, $i=1, \dots, n$, $X_0 = X_{n+1} = \#$ von links nach rechts solange, bis die erste Zeichenfolge $X_i X_{i+1}$ gefunden ist mit $X_i \triangleright X_{i+1}$. X_i ist dann das Ende der Primphrase.
2. Suche, ausgehend von X_i , nach links solange, bis die erste Zeichenfolge $X_{j-1} X_j$ gefunden ist mit $X_{j-1} \triangleleft X_j$. X_j ist dann der Anfang der Primphrase.
3. Prüfe, ob $X_j \dots X_i$ eine reduzierbare Teilzeichenfolge, d.h. eine rechte Seite einer Produktion ist. Wenn ja, dann führe die Reduktion durch und kehre zu 1 zurück. Wenn nein, prüfe ob $X_j \dots X_i = X_i = S$ und $X_{j-1} = X_{i+1} = \#$ ist. In diesem Fall ist die Analyse erfolgreich beendet, im anderen Fall war die Eingabezeichenfolge kein Wort der Sprache.

1.3.2 Beschreibung mit Hilfe eines Programms

Syntax und Semantik des folgenden Programms sind angelehnt an die Programmiersprache PASCAL [Wir.Jen,74]. Die Notation von Deklarationen von Konstanten und Variablen erfolgt nur symbolhaft zur Erklärung ihrer Bedeutung. Zum Beispiel steht statt

```

s : array [0 ..og] of integer;
i : integer
nur s[i] ergänzt durch den Kommentar
{i-tes Kellerelement}
```



```

if leftpart(j,i,variab) then
begin
    i := j; s[i] := variab; goto 1
end
    else
    if j-1 ≠ 0 or i+1 ≠ 2 or S[1] ≠ axiom then error(k); {Reduktion (3)}
    write ('true');
end
    else error(k);
2:end.

```

Es ist unmittelbar klar, daß der Algorithmus Wirweb für jede einfache Präzedenzsprache ein Analysator ist. Auf einen formalen Beweis dieser Behauptung muß aus genannten Gründen verzichtet werden. Statt dessen wird im nächsten Abschnitt die Arbeitsweise des Analysators Wirweb mit Hilfe einer Abbildung beschrieben und die der obigen Behauptung entsprechende Behauptung formal bewiesen.

1.3.3 Beschreibung mit Hilfe von Analysefunktionen

Grundlegend für die Beschreibung von Analysatoren für einfache Präzedenzsprachen mit Hilfe von Abbildungen ist der Begriff der Konfiguration .

Definition 1.12

Sei $G = (N, \Sigma, P, S)$ eine EPG, dann heißt

1. $Akonf_G := \{(\#, u\#) \mid u \in \Sigma^+\}$ die Menge der Anfangskonfigurationen über G.
2. $Zkonf_G := \{(\#\alpha, u\#) \mid \alpha \in \Gamma^+, u \in \Sigma^*\}$, die Menge der Zwischenkonfigurationen über G.
3. $Ekonf := \{true\} \cup \mathbb{N}^0$ die Menge der Endekonfigurationen über G.
4. $Konf_G := Akonf_G \cup Zkonf_G \cup Ekonf$ die Menge der Konfigurationen über G
5. Sei $k \in Konf_G \setminus Ekonf$, $k = (\#\alpha, u\#)$, dann heißt die Zeichenfolge α der Kellerinhalt und das Zeichen $Anf(u\#)$ das aktuelle Zeichen der Konfiguration k.

Die Arbeitsweise des Algorithmus Wirweb kann wie folgt beschrieben werden: Ausgehend von der Anfangskonfiguration $k_0 := (\#, a_1 \dots a_n \#)$ bildet solange Konfigurationen auf Konfigurationen ab, bis eine Endekonfiguration erreicht ist. Dabei repräsentiert die linke Komponente der aktuellen Konfiguration den Keller des Analysators.

Definition 1.13

Sei $G = (N, \Sigma, P, S)$ eine EPG, Konf_G die Menge der Konfigurationen über G . Eine Abbildung $F : \text{Konf}_G \rightarrow \text{Konf}_G$ heißt eine Analysefunktion für $L = L(G)$ genau dann, wenn gilt:

Zu jedem $u \in \Sigma^+$ existiert eine eindeutig bestimmte Folge $k_i \in \text{Konf}_G$, $i = 0, \dots, p$ von Konfigurationen mit den Eigenschaften:

1. $k_0 = (\#, u\#)$
2. $k_{i+1} = F(k_i) \quad i = 0, \dots, p-1$
3. $k_p = \begin{cases} \text{true} & \text{falls } u \in L(G) \\ |\tilde{u}\#|-1 & \text{falls } u \notin L(G) \wedge k_{p-1} = (\# \alpha, \tilde{u}\#) \end{cases}$

Die Folge $k_0, \dots, k_p$ heißt eine Analyse von u mit Hilfe von F . Jede Konfiguration k_i einer Analyse von u heißt eine mittels F definierte Konfiguration von u .

Jede Analysefunktion F für L definiert auf natürliche Weise einen Analysator für L .

Satz 1.1

Sei $G = (N, \Sigma, P, S)$ eine EPG, Konf_G die Menge der Konfigurationen über G , $F : \text{Konf}_G \rightarrow \text{Konf}_G$ eine Analysefunktion für $L = L(G)$, dann gilt:
Die Abbildung $A_F : \Sigma^+ \rightarrow \text{Ekonf}$ definiert durch:

$$A_F(u) := \begin{cases} \text{true} & \text{falls } (\#, u\#) F^+ \text{true} \quad (1) \\ |u|+1-k & \text{falls } (\#, u\#) F^+ k \end{cases}$$

ist ein Analysator für L .

(1) F kann aufgefaßt werden als Relation über Konf , d.h. F^+ ist die transitive Hülle von F .

Der Beweis von Satz 1.1 ist eine unmittelbare Folge aus Definition 1.13.

Definition 1.14

Seien die Voraussetzungen aus Satz 1.1 gegeben. Dann heißt der dort definierte Analysator A_P , der von F induzierte Analysator für L.

Die im folgenden Satz definierte Analysefunktion $F1$ induziert einen Analysator A_{F1} , der dem Algorithmus Wirweb entspricht. Diese Behauptung ist eine unmittelbare Folgerung aus der Definition von $F1$. Dennoch kann sie aus genannten Gründen nicht formal bewiesen werden.

Satz 1.2

Sei $G = (N, \Sigma, P, S)$ eine EPG, $Konf_G$ die Menge der Konfigurationen über G, dann gilt:

Die Abbildung $F1 : Konf_G \rightarrow Konf_G$ definiert durch:

$$F1(\#X_1X_2\dots X_i, u\#) := \left\{ \begin{array}{ll} (\#X_1\dots X_i, a, \tilde{u}\#) \text{ falls} & (1) \\ \quad \begin{array}{l} 1. u = a\tilde{u}, \tilde{u} \in \Sigma^*, a \in \Sigma \\ 2. X_i \dot{=} a \vee X_i \dot{<} a \end{array} & \\ (\#X_1\dots X_P, A, u\#) \text{ falls} & (2) \\ \quad \begin{array}{l} 1. X_i \dot{>} Anf(u\#) \\ 2. \exists p \in \mathbb{N}^0: p = \max_{m=q, \dots, i-1} (X_m \dot{<} X_{m+1}) \end{array} & \\ \text{true} & \\ \text{falls} & (3) \\ \quad i = 1 \wedge X_1 = S \wedge u = \epsilon & \\ |u\#| - 1 & \text{falls} & (4) \\ \quad X_i \dot{?} Anf(u\#) & \\ |u\#| - 1 & \text{falls} & (5) \\ \quad \begin{array}{l} 1. X_i \dot{>} Anf(u\#) \\ 2. \exists p \in \mathbb{N}^0: p = \max_{m=q, \dots, i-1} (X_m \dot{<} X_{m+1}) \end{array} & \\ \text{falls} & \\ \quad \begin{array}{l} 3. \nexists A \rightarrow X_{p+1} \dots X_i \in P \end{array} & \\ |u\#| - 1 & \text{falls} & (6) \\ \quad \begin{array}{l} 1. X_i \dot{>} Anf(u\#) \\ 2. \nexists m \in \mathbb{N}_{i-1}^0: X_m \dot{<} X_{m+1} \end{array} & \end{array} \right.$$

, wo $X_j \in \Gamma$ für $j = 1, \dots, i$, $X_0 := \#$, $u \in \Sigma^*$ gilt, ist eine Analysefunktion für $L = L(G)$.

Bemerkung 1.3

Die Fälle (4) - (6) der Definition von $F!$ repräsentieren die Fehlererkennung der Analysefunktion $F!$. Die in der Literatur üblichen Namen für die dabei erkannten Fehlertypen sind:

1. Zeichenpaarfehler (Character-Pair-Error) für (4)
2. Reduktionsfehler (Reduction-Error) für (5)
3. Typ1-Fehler (Type1-Error) für (6)

Im Beweis von Satz 1.2 werden zwei Hilfsätze benutzt, die dem Buch von Aho und Ullman [Aho.Ull,72] entnommen sind.

Hilfssatz 1.1

Sei $G = (N, \Sigma, P, S)$ eine EPG, dann gilt für $u \in \Sigma^*$:

$u \in L(G) \iff \exists$ genau eine Rechtsableitung $\omega_0, \omega_1, \dots, \omega_k$ von u .

Hilfssatz 1.2

Sei $G = (N, \Sigma, P, S)$ eine EPG, $X_0 X_1 \dots X_p \in \# \Gamma^*$, $u \in \Sigma^*$, $A \rightarrow X_{p+1} \dots X_q \in P$ mit

$$\# S \# \xRightarrow{R} X_0 \dots X_p A u \# \xRightarrow{R} X_0 \dots X_p X_{p+1} \dots X_q u \#$$

, dann gelten genau die folgenden Präzedenzrelationen:

1. Entweder $X_j \dot{=} X_{j+1}$ oder $X_j \triangleleft X_{j+1}$ für $j = 0, \dots, p-1$
2. $X_p \triangleleft X_{p+1}$
3. $X_j \dot{=} X_{j+1}$ für $j = p+1, \dots, q-1$
4. $X_q \triangleright \text{Anf}(u\#)$

Beweis von Satz 1.2:

Die Abbildung $F! : \text{Konf}_G \rightarrow \text{Konf}_G$ ist für alle Elemente aus $\text{Konf}_G \setminus \text{Ekonf}$ definiert. Nach endlich vielen Abbildungsschritten bildet $F!$ jedes Element von $\text{Konf}_G \setminus \text{Ekonf}$ auf ein Element von Ekonf ab. Dies folgt daraus,

daß in G keine Ableitung der Art $A \Rightarrow^+ A$ und keine ϵ -Produktion existiert. Deshalb ist es für die Richtigkeit von Satz 1.2 hinreichend zu zeigen, daß für $u \in \Sigma^+$ die folgende Äquivalenz gilt.

$$u \in L(G) \iff \exists k_i \in \text{Konf}_G, \quad i = 0, \dots, p \quad \text{mit:}$$

1. $k_0 = (\#, u\#)$
2. $k_{i+1} = F(k_i) \quad \text{für } i = 0, \dots, p-1$
3. $k_p = \text{true}$

" } "

Zuerst beweisen wir durch vollständige Induktion über i die folgende Hilfsbehauptung:

Hilfsbehauptung 1

Sei $\omega_0, \omega_1, \dots, \omega_i$ eine Rechtsableitung über G mit den Eigenschaften:

1. $\omega_0 = \# S \#$
2. $\omega_j \xrightarrow{R} \omega_{j+1} \quad \text{für } j = 0, \dots, i-1$
3. $\omega_{i-1} = X_0 X_1 \dots X_p A u \#$

^

$$\omega_i = X_0 X_1 \dots X_p X_{p+1} \dots X_q u \#, \quad \text{wo } X_0 = \#, X_j \in \Gamma, j=1, \dots, q,$$

$$A \rightarrow X_{p+1} \dots X_q \in P, u \in \Sigma^*$$

Dann existiert eine Folge $k_r, \dots, k_m$ von Konfigurationen über G mit den Eigenschaften:

1. $k_r := (X_0 \dots X_p X_{p+1} \dots X_q u \#)$
2. $k_{j+1} = F(k_j) \quad \text{für } j = r, \dots, m-1$
3. $k_m = (\# S, \#)$

Beweis durch vollständige Induktion über i

Induktionsanfang: $i = 1$

Für $i = 1$ gilt: $\omega_0 = \# S \#$ und $\omega_1 = \# X_1 \dots X_q \#$, dann existiert wegen $\omega_0 \xrightarrow{R} \omega_1$ eine Produktion $S \rightarrow X_1 \dots X_q$ in P.

Nach Hilfssatz 1.2 gelten die Relationen:

1. $\# \triangleleft X_1$
2. $X_j \doteq X_{j+1}$
3. $X_q > \#$

Nach Definition von Fl gilt dann für $k_1 := (\# X_0 \dots X_q, \#)$ und $k_2 := (\# S, \#)$

$$Fl(k_1) = k_2,$$

d.h. der Induktionsanfang ist bewiesen.

Induktionsschritt:

Die Behauptung ist für $i = n+1$ zu zeigen, unter der Voraussetzung, daß sie für $i \leq n$ gilt.

Beweis:

Sei also $\omega_0, \omega_1, \dots, \omega_{n+1}$ eine Rechtsableitung mit:

1. $\omega_0 = \# S \#$
2. $\omega_j \xrightarrow{R} \omega_{j+1}, j = 0, \dots, n-1$
3. $\omega_n = X_0 \dots X_p A u \#$
4. $\omega_{n+1} = X_0 \dots X_p X_{p+1} \dots X_q u \#$

dann gilt für $k_1 := (X_0 \dots X_p X_{p+1} \dots X_q, u \#)$ und $k_2 := (X_0 \dots X_p A, u \#)$

$$k_2 = Fl(k_1)$$

Da $\omega_0, \dots, \omega_n$ eine Rechtsableitung ist, hat ω_{n-1} die Form

$$\omega_{n-1} = X_0 \dots X_h B u' \# \quad \text{mit:}$$

$$B \rightarrow X_{h+1} \dots X_p A \quad u'' \in P \wedge u = u'' u'$$

Jetzt unterscheiden wir zwei Fälle:

1. $B \rightarrow X_{h+1} \dots X_p A \quad \text{d.h. } u'' = \varepsilon$
2. $B \rightarrow X_{h+1} \dots X_p A u'' \quad \text{d.h. } u'' \neq \varepsilon$

Für den Fall 1 folgt die Induktionsbehauptung direkt aus der Induktionsvoraussetzung, denn es existiert eine Folge $k_2, \dots, k_m$ mit den Eigenschaften:

1. $k_2 = (X_0 \dots X_p A, u \#)$
2. $k_j = Fl(k_{j-1}), \quad j=3, \dots, m$
3. $k_m = (\# S, \#)$

d.h. die Folge $k_1, k_2, \dots, k_m$ genügt den in der Induktionsbehauptung geforderten Eigenschaften.

Im Fall 2 sei $u'' = u_1 \dots u_r$ dann gilt:

1. $A \stackrel{*}{=} u_1$
2. $u_j \stackrel{*}{=} u_{j+1} \quad \text{für } j = 1, \dots, r-1$
3. $End(u'') = u_r \triangleright Anf(u' \#)$

d.h. für die Konfigurationenfolge $k_2, \dots, k_{r+1}$ definiert durch

$$\begin{aligned} k_2 &:= (X_0 \dots X_p A, u \#) \\ k_3 &:= (X_0 \dots X_p A u_1, u_2 \dots u_r, u' \#) \\ &\vdots \\ k_{r+2} &:= (X_0 \dots X_p A u_1 \dots u_r, u' \#) \end{aligned}$$

gilt

$$k_{j+1} = Fl(k_j) \quad \text{für } j = 2, \dots, r.$$

Nach Induktionsvoraussetzung existiert eine Folge

$k_{r+2}, \dots, k_m$ mit:

1. $k_{r+2} = (X_o \dots X_p \text{ Au}', u' \#)$
2. $k_{j+1} = Fl(k_j)$ für $j = r+2, \dots, m-1$
3. $k_m = (\# S, \#)$.

Daraus folgt: Die Folge

$k_1, \dots, k_{r+1}, k_{r+2}, \dots, k_m$

erfüllt die in der Induktionsbehauptung zu zeigenden Eigenschaften.

q.e.d (Hilfsbehauptung 1)

Aus der Hilfsbehauptung 1 kann die Richtung " $\rangle$ " des Satzes 2.1 leicht gefolgert werden.

Sei $u = u_1 \dots u_n \in L(G)$, dann existiert nach Hilfssatz 1.1 eine eindeutige bestimmte Rechtsableitung

$\omega_o, \omega_1, \dots, \omega_k$

von u bzgl. G , für welche gilt:

1. $\omega_o = \# S \#$
2. $\omega_i \in \# \Gamma^* \#$, $i = 1, \dots, k$
3. $\omega_{k-1} = \# u_1 \dots u_h \text{ Au}' \#$
4. $\omega_k = \# u_1 \dots u_h u_{h+1} \dots u_r u' \#$ wo $A \rightarrow u_{h+1} \dots u_r \in P$ und
 $u_1 \dots u_h u_{h+1} \dots u_r u' = u$

D.h. mit $\tilde{u} := u_{h+1} \dots u_r$ gilt für die Konfigurationenfolge

$k_o := (\#, u_1 \dots u_h \tilde{u} u' \#)$
 $k_1 := (\# u_1, u_2 \dots u_h \tilde{u} u' \#)$
 $\vdots$

$$\begin{aligned} k_h &:= (\# u_1 \dots u_h, u_{h+1} \dots u' \#) \\ &\vdots \\ k_r &:= (\# u_1 \dots u_h u_{h+1} \dots u_r, u' \#) \end{aligned}$$

nach Hilfssatz 1.2 und Definition von Fl

$$k_{j+1} = Fl(k_j) \quad \text{für } j = 0, \dots, r$$

Nach der Hilfsbehauptung 1 existiert eine Folge von Konfigurationen

$$k_{r+1}, \dots, k_m \quad \text{mit}$$

1. $k_{r+1} := (\# u_1 \dots u_r, u' \#)$
2. $k_{i+1} = Fl(k_i) \quad \text{für } i = 1, \dots, m-1$
3. $k_m = (\# S, \#)$

Daraus folgt, daß mit $k_{m+1} := true$ die Folge $k_0, \dots, k_{m+1}$ die gesuchte Folge ist.

q.e.d. (" } ")

"<"

Zuerst beweisen wir durch vollständige Induktion über i die folgende Hilfsbehauptung 2.

Hilfsbehauptung 2

Sei $k_0, \dots, k_i$, $i > 0$ eine Konfigurationenfolge mit den Eigenschaften:

1. $k_0 = (\# \alpha, u \#)$ mit : $\alpha \in \Gamma^* \wedge u \in \Sigma^* \wedge \alpha u \neq \varepsilon$
2. $k_{j+1} = Fl(k_j) \quad \text{für } j = 0, \dots, i-1$
3. $k_i = (\# S, \#)$

, dann ist $\bar{k}_0 := \alpha u$ eine Rechtssatzform von G .

Beweis durch vollständige Induktion über i

Induktionsanfang: $i = 1$

Es gilt: $k_0 = (\# \alpha, u \#) \wedge k_1 = (\# S, \#) \quad \left. \vphantom{k_0} \right\} \text{ (nach Def. von F1)}$

$u = \varepsilon \quad \text{und} \quad \alpha = X_1 \dots X_p \quad \text{mit} \quad S \rightarrow X_1 \dots X_p \in P \quad \left. \vphantom{u} \right\}$
 $\bar{k}_0 = X_1 \dots X_p \quad \text{ist eine Rechtssatzform über G.}$

Induktionsschritt:

Die Behauptung ist für $i = n+1$ zu zeigen, unter der Voraussetzung, daß sie für $i \leq n$ gilt.

Beweis

Sei $k_0, k_1, k_2, \dots, k_n, k_{n+1}$ eine Konfigurationenfolge der Länge $n+2$ mit den Eigenschaften 1 - 3 der Hilfsbehauptung 2. Dann erfüllt $k_1, \dots, k_{n+1}$ die Induktionsvoraussetzung und es gilt $\bar{k}_1 := \alpha_1 u_1$ ist Rechtssatzform von G.

Sei $\alpha_1 = X_1^1 \dots X_1^{j_1}$ und $k_0 = (\# X_0^1 \dots X_0^{j_0}, u_0 \#) =: (\# \alpha_0, u_0 \#)$ dann sind wegen $k_1 = F1(k_0)$ und $k_1 \notin \text{Konf}$ folgende Fälle möglich:

1. $X_0^{j_0} \doteq \text{Anf}(u_0 \#)$
2. $X_0^{j_0} \triangleleft \text{Anf}(u_0 \#)$
3. $X_0^{j_0} \triangleright \text{Anf}(u_0 \#)$

In den Fällen 1 und 2 gilt $\alpha_0 u_0 = \alpha_1 u_1$, d.h. da nach Induktionsvoraussetzung $\alpha_1 u_1$ eine Rechtssatzform ist, ist auch $\alpha_0 u_0$ eine Rechtssatzform.

Im Fall 3 gilt:

1. $u_0 = u_1$
2. $\exists q \in \{0, \dots, j_0 - 1\}$ mit:
 $q = j_1 - 1$
 $\wedge \exists X_1^{j_1} \rightarrow X_0^{q+1} \dots X_0^{j_0} \in P$

Da $\alpha_1 u_1$ nach Induktionsvoraussetzung eine Rechtssatzform mit $u_1 \in \Sigma^*$ ist, folgt, daß auch $\alpha_0 u_0$ eine Rechtssatzform ist, da wegen 2. (aus Fall 3)

$$\alpha_1 u_1 \xrightarrow{R} \alpha_0 u_0$$

gilt.

q.e.d (Hilfsbehauptung 2)

Aus der Hilfsbehauptung 2 kann die Richtung " $\Leftarrow$ " der für die Richtigkeit des Satzes 2.1 zu zeigenden Behauptung leicht gefolgert werden.

Sei $k_0 \dots k_p, p \geq 2$ eine Folge von Konfigurationen über G mit den Eigenschaften:

1. $k_0 = (\#, u \#)$
2. $k_{i+1} = Fl(k_i)$ für $i = 0, \dots, p-1$
3. $k_p = \text{true}$.

Nach Definition von Fl gilt: $k_{p-1} = (\# S, \#)$, das heißt, aus der Hilfsbehauptung 2 folgt, daß u eine Rechtssatzform von G ist, d.h. daß $u \in L(G)$ ist.

q.e.d (" $\Leftarrow$ ")

q.e.d. (Satz 1.2)

Bemerkung 1.4

Die Anwendbarkeit der in diesem Abschnitt entwickelten Beschreibungsmethode ist nicht beschränkt auf Analysatoren für einfache Präzedenzsprachen. Nach entsprechenden Modifikationen des Begriffs der Konfiguration können zu allen Bottom-Up Analysatoren Analysefunktionen definiert werden, für die ein, zu Satz 1.2 analoger Satz gilt.

2. DIE ERKENNUNG VON SYNTAXFEHLERN MIT HILFE VON ANALYSATOREN FÜR EINFACHE PRÄZEDENZSPRACHEN

Eine Grundvoraussetzung für jede Fehlerbehandlung ist die Fehlererkennung. Die bisher bekannten Analysealgorithmen für einfache Präzedenzsprachen erkennen i.a. die Existenz von Syntaxfehlern nicht frühestmöglich. Dadurch wird die Behandlung von Syntaxfehlern erschwert bzw. verschlechtert. Deshalb wird in diesem Abschnitt ein Analysator für einfache Präzedenzsprachen konstruiert, der die Existenz von Syntaxfehlern so früh wie möglich erkennt.

2.1 Zur frühestmöglichen Erkennung von Syntaxfehlern

Ein Analysator für eine Sprache L , welcher Eingabezeichenfolgen von links nach rechts abarbeitet ⁽¹⁾, kann die Existenz eines Syntaxfehlers in dem zu analysierenden Wort u , frühestens genau dann erkennen, wenn der bereits gelesene Teil der Eingabezeichenfolge konkateniert mit den nächsten zu verarbeiteten Zeichen kein Präfix von L ist. Dieser Teil der Eingabezeichenfolge bezeichnet man als das längste Präfix der Zeichenfolge u .

Definition 2.1

Sei L eine Sprache über dem Alphabet Σ und $u, x \in \Sigma^*$, dann gilt:

- u heißt das längste Präfix von x (bzgl. L): 
1. $u \in \text{Präfix}(x) \wedge u \in \text{Präfix}(L)$
 2. Sei $v \in \Sigma^*$ mit: $v \in \text{Präfix}(x) \wedge v \in \text{Präfix}(L)$
dann gilt: $v \in \text{Präfix}(u)$

Die Eigenschaft von Analysatoren, die Existenz von Fehlern genau dann zu erkennen, wenn der bereits analysierte Teil der Eingabezeichenfolge

(1) Alle in dieser Arbeit diskutierten Analysatoren lesen von links nach rechts, ohne daß dies im weiteren explizit erwähnt wird.

kein Präfix der Sprache ist, wollen wir als die LP-Eigenschaft bezeichnen.

Definition 2.2

Sei $G = (N, \Sigma, P, S)$ eine KFG, A ein Analysator für $L = L(G)$, dann gilt:

A hat die LP-Eigenschaft für L : $\forall$

$\forall u \in \Sigma^+ \setminus L$ gilt : $A(u) = k, k \in \mathbb{N}$ $\forall$ das längste Präfix u von x bzgl. L hat die Länge $k-1$

Die LP-Eigenschaft ist für Übersetzer von Programmiersprachen aus vielen Gründen wichtig. Einige davon sind: Die Möglichkeit zutreffende Fehlerdiagnosen zu stellen; die Möglichkeit Syntaxfehler ohne Zurücksetzen des Eingabezeigers zu korrigieren; Codeerzeugung für fehlerhafte Zeichenfolgen zu vermeiden; möglichst viele Fehler zu erkennen. Das zuletzt genannte Problem wird bei den im Abschnitt 3 konstruierten Fehlerbehandlungsalgorithmen eine wichtige Rolle spielen. In den folgenden drei Abschnitten wird gezeigt werden, daß die bisher bekannten Analysealgorithmen für einfache Präzedenzsprachen die LP-Eigenschaft nicht haben.

2.2 Die Fehlererkennung der bisher bekannten Analysatoren

2.2.1 Der Analysator von Wirth - Weber

Wie im vorigen Abschnitt bereits erwähnt, wird die Arbeitsweise des Analysators von Wirth-Weber [Wir,Web.66] durch die Abbildung F_1 beschrieben. Er erkennt die Existenz von Syntaxfehlern in genau drei verschiedenen Situationen der Syntaxanalyse. Der Zeichenpaarfehler tritt auf, wenn zwischen dem obersten Kellerelement und dem nächsten Eingabezeichen keine Präzedenzrelation besteht.

Beispiel 2.1: Zeichenpaarfehler

Sei das bezüglich L1 fehlerhafte Programm

```
program id;  
begin  
    id :: id + id  
end.
```

gegeben. Der Fehler sei durch Ersetzen von = durch : entstanden.

Bei der Analyse der obigen Eingabezeichenfolge mit Hilfe von F1 bzgl. G1 ergibt sich die Konfiguration

$(\#Ph \text{ begin id: } , :id + id \text{ end. } \#)$

Da zwischen : und : keine Relation besteht, wird diese Konfiguration auf $|:id + id \text{ end. } \#|-1$ abgebildet.

Δ

Ein Typ1-Fehler tritt auf, wenn zu bereits gefundenem Ende einer Primphrase kein möglicher Anfang im Keller steht. Dies kann nur dann vorkommen, wenn zwischen dem ersten (#) und zweiten Zeichen des Kellers keine Relation besteht. Bei der Sprache L1 definiert durch G1 kann dieser Fehlertyp nicht auftreten, wie leicht gezeigt werden kann.

Ein Reduktionsfehler tritt auf, wenn nach gefundenem Ende und Anfang von potentiellen Primphrasen sich herausstellt, daß der von Anfang und Ende eingeschlossene Teil keine rechte Seite einer Produktion ist.

Beispiel 2.2: Reduktionsfehler

Sei das bezüglich L1 fehlerhafte Programm

```
program id;  
begin  
    id [id]: id  
end.
```

gegeben.

Der Fehler sei durch Weglassen von = hinter : entstanden. Bei der Analyse der obigen Zeichenfolge mit Hilfe von F1, ergibt sich der Keller-

inhalt

$\#Ph \text{ begin } id [E]:$

Das aktuelle Zeichen ist id .

Es gelten die Relationen:

$$\begin{array}{rcl} : & \triangleright & id \\] & \doteq & : \\ E & \doteq &] \\ [& \doteq & E \\ id & \doteq &] \\ begin & \triangleleft & id \end{array}$$

Die Zeichenfolge $id [E]:$ ist aber keine rechte Seite der Grammatik G_1 , d.h. es gilt $F_1(\#Ph \text{ begin } id [E]:, id \text{ end.} \#) = 4$.

Δ

Es ist unmittelbar klar, daß A_{F_1} die LP-Eigenschaft nicht für alle Präzedenzsprachen haben kann, denn nach Ausführung einer Reduktion wird die entsprechende Variable gekellert, ohne daß geprüft wird, ob die zu kellernde Variable mit dem obersten Kellerelement eine der Relationen $\triangleleft$ oder $\doteq$ hat. Ist dies nicht der Fall, so tritt nach Fortsetzung der Analyse ein Typ1-Fehler oder ein Reduktionsfehler auf.

Beispiel 2.3: Zur Fehlererkennung von A_{F_1}

Sei das bezüglich L_1 fehlerhafte Programm

```
program id;
begin
    id : id
end.
```

gegeben. Das längste Präfix ist

```
program id; begin id :
```

Das aktuelle Zeichen ist id .

Der Kellereinhalt nach Analyse des längsten Präfixes ist

Ph begin id :

Wegen der Gültigkeit der Relationen:

begin $\Leftarrow$ *id*

id $\hat{=}$:

: $\triangleright$ *id*

wird *id* : auf *IL* reduziert. Im weiteren wird dann *IL* auf *IL1* reduziert und wegen *Ph* $\Leftarrow$ *begin* und *id* $\triangleright$ *end* wird beim Versuch der Reduktion von *begin IL1 id* ein Reduktionsfehler entdeckt. Δ

Leinius [Lei.70] hat diese Schwäche von A_{F1} erkannt und *F1* so erweitert, daß dieser Fehlertyp früher erkannt wird.

2.2.2 Der Algorithmus von Leinius

Leinius vermeidet die Schwäche von A_{F1} , dadurch, daß er, bevor er eine durch Reduktion gewonnene Variable kellert, überprüft, ob eine der Relationen $\Leftarrow$ oder $\hat{=}$ zwischen dem obersten Kellerelement und der zu kellernden Variablen gilt. ⁽¹⁾ Diese Prüfung bezeichnen wir als Prüfung auf Kellerbarkeit, den entsprechenden Fehlertyp als Kellerungsfehler.

Die um die Prüfung auf Kellerbarkeit erweiterte Analysefunktion *F1* bezeichnen wir mit *F2*.

Satz 2.1

Sei $G = (N, \Sigma, P, S)$ eine EPG, Konf_G die Menge der Konfigurationen über *G*, dann gilt:

Die Abbildung $F2: \text{Konf}_G \rightarrow \text{Konf}_G$ definiert durch:

(1)

Die Relation $\triangleright$ kann aufgrund der Definition der Präzedenzrelationen nicht auftreten.

$$F2(\#X_1, \dots, X_i, u\#) := \begin{cases} |u\#|-1 & \text{falls} \\ 1. X_i \triangleright \text{Anf}(u\#) \\ 2. \exists p \in \mathbb{N} \text{ mit } p = \max_{m=0, \dots, i-1} (X_m \triangleleft X_{m+1}) \\ 3. \exists \text{ Produktion} \\ A \rightarrow X_{m+1} \dots X_i \in P \\ 4. X_p \triangleright A \\ F1(\#X_1, \dots, X_i, u\#) & \text{sonst (Fälle (1)-(6) aus} \\ & \text{Satz 1.2)} \end{cases} \quad (7)$$

,wo $X_j \in \Gamma$ für $j=1, \dots, i$, $X_0 = \#$, $u \in \Sigma^*$

gilt, ist eine Analysefunktion für $L = L(G)$.

F2 unterscheidet sich von F1 dadurch, daß nach einer Reduktion die gewonnene Variable nicht sofort gekellert, sondern wie ein Eingabezeichen behandelt wird. Dadurch wird vor der Kellierung noch auf Präzedenzrelationen geprüft (Fall (7)). Deswegen ist der Beweis von Satz 2.1 eine triviale Erweiterung des Beweises von Satz 1.2.

Der von F2 induzierte Analysator A_{F2} erkennt den Fehler aus Beispiel 2.3 dort, wo das längste Präfix zu Ende ist, da das oberste Kellerelement *begin* zur Variablen *IL* keine Präzedenzrelation hat. Dennoch gibt es Beispiele, in denen auch A_{F2} das längste Präfix nicht erkennt.

Beispiel 2.4: Zur Fehlererkennung von A_{F2}

Sei das bezüglich L1 fehlerhafte Programm

```
program id := id + id;
begin
    id := id
end.
```

gegeben. Das längste Präfix ist *program id*. F2 erkennt den Fehler erst nach Lesen von ; und dem Versuch der Reduktion der Zeichenfolge

$program\ id := id + id \ .$

Δ

Der Grund für die verspätete Fehlererkennung ist die Existenz der Relation $\hat{=}$ zwischen id und $:$. Da F2 zur Entscheidung, ob ein Zeichen gekellert wird, nur die Relationen zwischen dem obersten Kellerelement und dem aktuellen Zeichen heranzieht, wird nicht erkannt, daß nach dem Schlüsselwort *program* keine Zuweisung kommen darf.

Rhodes [Rho.73] hat in seiner Arbeit eine Erweiterung von F2 vorgeschlagen, mit Hilfe welcher der Fehlertyp aus Beispiel 2.4 früher erkannt wird.

2.2.3 Der Algorithmus von Rhodes

Der im Beispiel 2.4 entdeckte Fehlertyp kann immer dann auftreten, wenn es ein Zeichen X gibt, welches in den rechten Seiten von verschiedenen Produktionen vorkommt. In diesem Fall ist es möglich, daß ein Zeichen gekellert wird, welches bewirkt, daß diejenige Teilzeichenfolge ⁽¹⁾ des Kellers, welche vom obersten Kellerelement, das mit seinem Vorgänger die Relation $\hat{=}$ hat und vom zuletzt gekellerten Zeichen begrenzt wird, kein Präfix einer Produktion mehr ist. In diesem Fall wird, wenn nicht zuvor ein weiterer Fehler entdeckt wird, bei einer der folgenden Reduktionsversuchen ein Fehler auftreten. Im Beispiel 2.4 wird die späte Fehlererkennung durch die Produktionen

$Pi \rightarrow program\ id \text{ und } A \rightarrow id : G\ E$

verursacht. Vor dem Lesen von $:$ ist *program id* das zu oberst im Keller stehende Präfix einer Produktion. id hat zu $:$ die Relation $\hat{=}$, das heißt $:$ wird gekellert. Die Zeichenfolge *program id :* ist jedoch kein Präfix einer Produktion. F2 erkennt dies erst bei dem Versuch der Reduktion von *program id : G E*. Um diese späte Fehlererkennung zu vermeiden, prüft der von Rhodes vorgeschlagene Algorithmus beim Auftreten

(1)

Wir bezeichnen diese Zeichenfolge im folgenden als aktuelles Präfix.

der Relation $\dot{=}$ vor der Kellierung ab, ob das aktuelle Präfix dadurch verlängert wird. Ist das nicht der Fall, so liegt ein Fehler, im folgenden mit RS-Fehler ⁽¹⁾ bezeichnet, vor. Die wesentliche Eigenschaft der von Rhodes vorgeschlagenen Implementierung ist, daß die Prüfung auf RS-Fehler so vorgenommen werden kann, daß kein zusätzlicher Laufzeitaufwand gegenüber dem Algorithmus von Leinius entsteht.

Seine Realisierung in Form einer Analysefunktion ergibt sich durch Erweiterung von F2 zu F3.

Satz 2.2

Sei $G = (N, \Sigma, P, S)$ eine EPG, Konf_G die Menge der Konfigurationen über G , dann gilt:

Die Abbildung $F3: \text{Konf}_G \rightarrow \text{Konf}_G$ definiert durch:

$$F3(\#X_1 \dots X_i, u\#) := \begin{cases} |u\#|-1 & \begin{array}{l} \text{falls} \\ 1. a = \text{Anf}(u\#) \\ 2. X_i \dot{=} a \\ 3. \exists p = \max_{m=0, \dots, i-1} (X_m \triangleleft X_{m+1}) \\ 4. \nexists A \rightarrow X_{p+1} \dots X_i a \beta \in P, \beta \in \Gamma^* \end{array} & (8) \\ |u\#|-1 & \begin{array}{l} \text{falls} \\ 1. X_i \triangleright \text{Anf}(u\#) \\ 2. \exists p = \max_{m=0, \dots, i-1} (X_m \triangleleft X_{m+1}) \\ 3. \exists A \rightarrow X_{p+1} \dots X_i \in P \\ 4. X_i \dot{=} A \\ 5. \exists^p q = \max_{m=0, \dots, p-1} (X_m \triangleleft X_{m+1}) \\ 6. \nexists B \rightarrow X_{q+1} \dots X_p A \beta \in P, \beta \in \Gamma^* \end{array} & (9) \\ F1(\#X_1 \dots X_i, u\#) & \text{sonst (Fälle (1)-(7) aus Satz 2.1)} \end{cases}$$

, wo $X_j \in \Gamma$ für $j = 1, \dots, i$, $X_0 = \#$, $u \in \Sigma^*$ gilt, ist eine Analysefunktion für $L = L(G)$.

(1) Rhodes bezeichnet diesen Fehler mit RHS-Error (Right-Hand-Side-Error).

F3 unterscheidet sich von F2 nur durch die Prüfung auf RS-Fehler (Fälle (8), (9)). Deswegen ist der Beweis von Satz 2.2 ebenfalls eine triviale Erweiterung des Beweises von Satz 1.2. Der durch F3 induzierte Analysator A_{F3} erkennt den im Beispiel 2.4 demonstrierten Fehlertyp dort, wo das längste Präfix zu Ende ist. Im Beispiel 2.4 ist das nach dem Lesen von : .

Ferner hat F3 die Eigenschaft, daß bei der Analyse einer Zeichenfolge u nur Kellerinhalte auftreten, welche als Folge von Präfixen von Produktionen aufgefaßt werden können.

Satz 2.3

Sei $G = (N, \Sigma, P, S)$ eine EPG, $u \in \Sigma^*$ und $k_0, \dots, k_p$ eine Analyse von u mit Hilfe von F3, dann gilt für $k \in \{k_0, \dots, k_{p-1}\}$:

Der Kellerinhalt der Konfiguration

$$k = (X_0 \dots X_i, u\#)$$

läßt sich darstellen in der Form

$$\psi_0 \dots \psi_r \quad \text{wo gilt:}$$

1. $\psi_0 = X_0 = \#$
2. $\psi_j = X_{p_j} X_{p_j+1} \dots X_{p_{j+1}-1}$ mit : $X_m = X_{m+1}$ für $m \in \{p_j, \dots, p_{j+1}-2\}$
für $j = 1, \dots, r$
3. Zu $\psi_j \quad \exists A_j \rightarrow \psi_j \beta_j \in P$ für $j = 1, \dots, r$.
4. $X_{p_{j-1}-1} \leftrightarrow X_{p_j}$ für $j = 1, \dots, r$

Die Richtigkeit des Satzes 2.3 ist eine unmittelbare Folgerung aus der Definition von F3 in Satz 2.2. Durch die dort festgelegte Prüfung auf rechte Seiten wird erreicht, daß sämtliche ψ_j Präfixe von rechten Seiten sind.

Definition 2.3

Seien die Voraussetzungen von Satz 2.3 gegeben, dann gilt:

1. $\psi_0 \dots \psi_r$ heißt eine Präfixdarstellung des durch $k = (X_0 \dots X_i, u \#)$ definierten Kellerinhalts
2. ψ_r heißt das aktuelle Präfix von k .

Auch A_{F3} erkennt nicht alle Fehler genau dann, wenn das längste Präfix zu Ende ist.

Beispiel 2.5

Sei das bezüglich L1 fehlerhafte Programm

```

program id ;
begin
    if id := id + id + id then id := id
end .

```

gegeben. Das längste Präfix endet mit id vor dem ersten $:$. Erkennt wird die Existenz eines Fehlers erst nach der Reduktion von $id := id + id + id$ auf A . Der Grund dafür ist, daß es in G1 mehr als eine Variable V gibt, mit $id \in \text{Anf}^+ V$. Da die Analysefunktion $F3$ nur prüft, ob id : Präfix irgend einer Produktion ist, wird nicht erkannt, daß die Zeichenfolge id : kein Präfix einer aus E ableitbaren Rechtssatzform ist. Δ

Rhodes hat in seiner Arbeit darauf hingewiesen, daß auch sein Algorithmus (A_{F3}) die LP-Eigenschaft nicht besitzt, jedoch keine weiteren Anstrengungen unternommen, dies zu beheben.

Deswegen ist es das Ziel des nächsten Abschnitts alle Fälle zu bestimmen, in denen A_{F3} das längste Präfix nicht erkennt und daraufhin einen Algorithmus zu entwickeln, der die LP-Eigenschaft besitzt.

2.3 Entwurf eines Analysators für einfache Präzedenzsprachen, welcher die Existenz von Syntaxfehlern frühestmöglich erkennt.

2.3.1 Anschauliche Darstellung aller Fälle, in denen der Algorithmus von Rhodes Fehler nicht frühestmöglich erkennt.

Bei der Eingabezeichenfolge aus Beispiel 2.5 wurde die späte Fehlererkennung durch die Analysefunktion F3 dadurch verursacht, daß bei Kellerung eines Zeichens, welches zum obersten Kellerelement die Relation $\neq$ hat, nur geprüft wurde, ob das aktuelle Präfix dadurch verlängert wird, d.h. ob es eine Produktion gibt, für die das um das aktuelle Zeichen verlängerte aktuelle Präfix ein Präfix ihrer rechten Seite ist. Um die späte Fehlererkennung zu vermeiden, müßte jedoch geprüft werden, ob die entsprechende linke Seite zu dem bereits erstellten Kellerinhalt paßt, d.h. ob die linke Seite ableitbar ist aus der für das aktuelle Präfix gesuchten Variablen. Im Beispiel 2.5 war die gesuchte Variable E , die Zeichenfolge id : war nicht aus E ableitbar. Würde man zu jedem Zeitpunkt der Analyse die Menge der gesuchten Variablen kennen, so könnte man diesen Fehlertyp abprüfen.

Eine analoge Situation tritt auf, wenn zwischen dem obersten Kellerelement und dem aktuellen Zeichen die Relation $\neq$ gilt.

Beispiel 2.6: Zur Fehlererkennung von F3

Sei das bezüglich L1 fehlerhafte Programm

```
program id;  
var  
    id := id;  
begin  
    id := id  
end.
```

gegeben. Das längste Präfix endet beim ersten Auftreten von $=$. Der Kellerinhalt vor dem Lesen von $=$ ist $\#Ph \text{ var } id$:. Zwischen $:$ und $=$ gilt die Relation $\neq$, d.h. $=$ wird von F3 gekellert. Der Fehler wird erst er-

kannt nach der Reduktion von $id := id$ auf A .

Δ

Um im obigen Fall den Fehler früher erkennen zu können, müßte die Analysefunktion gespeichert haben, daß die Zeichenfolge id : innerhalb einer Variablendeklaration vorgekommen ist. Daraus könnte sie erkennen, daß die Zeichenfolge id : ein Präfix einer aus der Variablen Ill ableitbaren Zeichenfolge sein muß. Für diesen Sachverhalt gebrauchen wir wieder die Sprechweise, daß Ill eine der für das aktuelle Präfix gesuchten Variablen ist.

Gilt zwischen dem obersten Kellerelement und dem aktuellen Zeichen die Relation $\triangleright$, so ist es bei Verwendung der Analysefunktion F3 möglich, daß erst nach einigen Reduktionen erkannt wird, ob der bisher gelesene Teil der Eingabezeichenfolge konkateniert mit dem aktuellen Zeichen kein Präfix der Sprache mehr ist.

Beispiel 2.7: Zur Fehlererkennung von F3

Sei die Sprache $L2 := \{abc\} \cup \{(bf)^n e \mid n \in \mathbb{N}\} \cup \{abfd\}$ durch die einfache Präzedenzgrammatik $G2 = \{N2, \Sigma2, P2, S\}$ gegeben.

Die Menge $P2$ bestehe aus den Produktionen:

$$\begin{aligned} S &\rightarrow A \mid Be \mid H \\ A &\rightarrow aX \\ X &\rightarrow bY \\ Y &\rightarrow c \\ B &\rightarrow D \\ D &\rightarrow bE \\ E &\rightarrow fD \mid f \\ H &\rightarrow aI \\ I &\rightarrow bL \\ L &\rightarrow fd \end{aligned}$$

Die Matrix der Präzedenzrelationen ist:

	S	A	B	D	H	X	Y	E	I	L	a	b	c	f	d	e	#
S																	>
A																	>
B																	
D																>	>
H																>	
X																>	
Y																>	
E																>	>
I																>	
L																>	
a																	
b																	
c																	
f																	
d																	
e																	
#																	

Bei der Analyse der Eingabezeichenfolge $\#abfbfe\#$ gelten die Relationen:

$$\begin{aligned} \# &\prec a \\ a &\prec b \\ b &\prec f \\ f &\succ e \end{aligned}$$

Deshalb reduziert F3 das Zeichen f auf E , und wegen $b \doteq E$ die Zeichenfolge bE auf D . Erst jetzt wird ein Fehler erkannt. Δ

Um den im obigen Beispiel demonstrierten Fehler vor Ausführung der beiden Reduktionen erkennen zu können, müßte der Analysefunktion bekannt sein, daß nach der Zeichenfolge ab nur Zeichenfolgen gesucht werden, die aus einer der Variablen L oder Y abgeleitet werden können. Es wäre dann klar, daß das Zeichen f nicht auf E reduziert werden darf. D.h. die Variablen L, Y sind die für das aktuelle Präfix f gesuchten Variablen.

Es ist anschaulich unmittelbar klar, daß die in diesem Abschnitt besprochenen drei Fälle genau diejenigen Fälle sind, in denen F3 die Existenz von Syntaxfehlern später als wünschenswert erkennt. Um dies beweisen zu können, muß die Menge der gesuchten Variablen formal definiert werden.

2.3.2 Die Menge der gesuchten Variablen

In Satz 2.3 wurde gezeigt, daß jeder mit Hilfe von F3 definierte Kellerinhalt γ einer von den Endekonfigurationen verschiedenen Konfiguration als Folge von Präfixen ψ_i von Produktionen interpretiert werden kann. Ist der die Folge ψ_i definierende Kellerinhalt γ auch ein Präfix einer Rechtssatzform, so bezeichnen wir ihn als Bottom-Up Präfix. Die Folge der ψ_i bezeichnen wir als eine Präfixdarstellung von γ . Jedem Bottom-Up Präfix kann auf eindeutige Weise eine Menge von Variablen zugeordnet werden, die wir als Menge der für $\# \gamma$ gesuchten Variablen bezeichnen.

Definition 2.4

Sei $G = (N, \Sigma, P, S)$ eine KFG, dann gilt:

1. $\gamma \in \Gamma^+$ heißt ein Bottom-Up Präfix (BU-Präfix) (der Rechtssatzform $\alpha\beta w$) von G , : ✕

$$\exists s \xrightarrow{R^+} \alpha A w \Rightarrow \alpha \beta w \text{ mit } \alpha, \beta \in \Gamma^*, A \in N, w \in \Sigma^*$$

und $\gamma \in \text{Präfix}(\alpha\beta)$

2. ϵ heißt ein BU-Präfix.

In der obigen Definition bezeichnen wir γ deshalb als Bottom-Up Präfix, weil die Kellerinhalte, die bei der Bottom-Up Analyse eines fehlerfreien Wortes auftreten, stets Bottom-Up Präfixe sind. Dies deshalb, weil bei der Bottom-Up Analyse Zeichen genau solange gekellert werden, bis die obersten Kellerelemente die Primphrase der aus dem Kellerinhalt und dem noch zu analysierenden Teil der Eingabezeichenfolge gebildeten Satzform darstellen. Genau dann wird eine Reduktion durchgeführt.

Satz 2.4

Sei $G = (N, \Sigma, P, S)$ eine EPG, $\gamma \in \Gamma^*$ ein BU-Präfix von G , dann gilt:

Es existiert genau eine Folge $\psi_i \in \Gamma^* \cup \{\#\}$ $i=0, \dots, k$ mit den Eigenschaften:

1. $\psi_0 \psi_1 \dots \psi_k = \# \gamma$

2. $\psi_0 = \#$

3. Für $i = 1, \dots, k$ gilt:

$$\psi_i = X_1^i \dots X_{p_i}^i \quad \text{mit } X_q^i \in \Gamma \quad \text{für } q = 1, \dots, p_i,$$

$$\text{und } X_q^i = X_{q+1}^i \quad \text{für } q = 1, \dots, p_i - 1$$

$$4. \quad X_{p_i}^i \Leftarrow X_1^{i+1} \quad \text{für } i = 0, \dots, k-1$$

Der Beweis des Satzes ist eine unmittelbare Konsequenz des Hilfssatzes 1.2

Definition 2.5

Sei $G = (N, \Sigma, P, S)$ eine EPG, $\gamma \in \Gamma^*$ ein BU-Präfix, dann gilt:

Die nach Satz 2.4 definierte Folge ψ_i heißt die Präfixdarstellung von $\# \gamma$ (bzgl. G).

Bei der obigen Definition ist zu beachten, daß die Zeichenfolge ψ_i Präfixe von rechten Seiten sind, während die Zeichenfolge γ ein Präfix einer Rechtssatzform ist.

Satz 2.5

Sei $G = (N, \Sigma, P, S)$ eine EPG, $\gamma \in \Gamma^+$ ein BU-Präfix der Rechtssatzform $\alpha\beta w$, $\alpha, \beta \in \Gamma^*$, $w \in \Sigma^*$ über G . β sei Primphrase von $\alpha\beta w$, $\psi_0 \dots \psi_k$ sei eine Präfixdarstellung von $\# \gamma$ und ω_j , $j = 0, \dots, p$, die Rechtsableitung von $\alpha\beta w$, dann gilt:

Für $i = 1, \dots, k$ existieren Produktionen $A_i \rightarrow \psi_i \lambda_i \in P$ und Rechtssatzformen $\omega_{j_i} \in \{\omega_0, \dots, \omega_p\}$ mit der Eigenschaft:

$$\# \omega_{j_i} = \psi_0 \dots \psi_{i-1} A_i w_i$$

$$\wedge$$

$$\# \omega_{j_{i+1}} = \psi_0 \dots \psi_{i-1} \psi_i \lambda_i w_i$$

Beweis: Vollständige Induktion über k

Zunächst beweisen wir die folgende

Hilfsbehauptung:

Sei $G = (N, \Sigma, P, S)$ eine KFG, $\gamma \in \Gamma^+$ eine BU-Präfix der Rechtssatzform $\omega = \alpha\beta w$. $\beta \in \Gamma^+$ sei Primphrase von ω und $\psi_0 \dots \psi_k$ eine Präfixdarstellung von $\# \gamma$ mit $\psi_k \notin \text{Präfix}(\beta)$, dann gilt:

Es existiert eine Rechtssatzform $\bar{\omega}$ mit den Eigenschaften:

1. $\bar{\omega} \xrightarrow{R} \omega$
2. γ ist BU-Präfix von $\bar{\omega}$
3. $\psi_k \in \text{Präfix}(\bar{\beta})$, wenn $\bar{\beta}$ Primphrase von $\bar{\omega}$ ist.

Beweis der Hilfsbehauptung:

Ist ψ_k kein Präfix von β , dann erhält man durch Reduktion der Satzform ω eine Satzform ω_1 , für welche gilt:

1. $\omega_1 \xrightarrow{R} \omega$
2. γ ist BU-Präfix von ω_1

Ist β_1 die Primphrase von ω_1 und gilt $\psi_k \in \text{Präfix}(\beta_1)$, so ist die Behauptung bewiesen. Andernfalls erhält man durch Reduktion von β_1 eine Satzform ω_2 , welche die Eigenschaften 1 und 2 bzgl. ω_1 hat. Da nach endlich vielen Reduktionen ω auf S reduziert wird, muß es also eine Satzform $\bar{\omega}$ mit den gewünschten Eigenschaften geben.

q.e.d. (Hilfsbehauptung)

Ind.Anf.: $k = 1$

Sei $\# \gamma = \psi_0 \psi_1$, dann existiert zur Rechtssatzform $\omega := \alpha \beta \omega$ nach der Hilfsbehauptung eine Rechtssatzform $\bar{\omega} := \bar{\alpha} \bar{\beta} \bar{\omega}$ mit den Eigenschaften:

1. $\bar{\beta}$ ist Primphrase von $\bar{\omega} \wedge \bar{\omega} \xrightarrow{R} \omega$
2. $\gamma = \psi_1 \dots \psi_k$ ist BU-Präfix von $\bar{\omega}$
3. $\psi_1 \in \text{Präfix}(\bar{\beta})$

D.h. es existiert eine Produktion $A \rightarrow \bar{\beta}$ in P und es gilt:

$$\omega_{j_1+1} = \bar{\omega} = \bar{\beta} \bar{\omega}$$

und

$$\omega_{j_1} = A \bar{\omega}$$

Ind.Schritt:

Es ist zu zeigen, daß aus der Gültigkeit der Behauptung für $k \leq m$, ihre Gültigkeit für $k = m+1$ folgt.

Sei $\# \gamma = \psi_0 \psi_1 \dots \psi_{m+1}$ dann kann wegen der Hilfsbehauptung o.B.d.A. angenommen werden, daß gilt:

$$\alpha = \psi_1 \dots \psi_m \quad \text{und} \quad \psi_{m+1} \in \text{Präfix}(\beta)$$

Sei $A \rightarrow \psi_{m+1} \lambda_{m+1} = \beta$, $\lambda_{m+1} \in \Gamma^*$ dann gilt weiter:

$$\omega_{j_{m+1}} := \psi_1 \dots \psi_m \beta w$$

und

$$\omega_{j_{m+1}-1} := \psi_1 \dots \psi_m A w .$$

D.h. $\psi_1 \dots \psi_m A$ ist ein BU-Präfix von ω_{p-1} , da die Primphrase von ω_{p-1} keine Teilzeichenfolge von $\psi_1 \dots \psi_m A$ sein kann.

Jetzt sind zwei Fälle möglich:

1. $\text{End}(\psi_m) \doteq A$
2. $\text{End}(\psi_m) \triangleleft A$

Im ersten Fall ist

$$\psi_1 \dots \psi_{m-1} \tilde{\psi}_m \quad \text{mit} \quad \tilde{\psi}_m := \psi_m A$$

eine Präfixdarstellung der Länge m von $\psi_1 \dots \psi_m A$. Daraus folgt die Behauptung durch Anwendung der Induktionsvoraussetzung.

Im zweiten Fall gewinnt man stets durch endlich viele Reduktionen von $\psi_1 \dots \psi_m A w$ eine Variable $C \in N$ mit den Eigenschaften:

1. $\exists \omega_{j_1} \in \{\omega_0, \dots, \omega_{p-1}\}$ mit $\omega_{j_1} = \psi_1 \dots \psi_m C \lambda$, $\lambda \in \Gamma^*$
2. $\text{End}(\psi_m) \doteq C$
3. $C \lambda \xrightarrow{R} A w$

Dann ist

$$\psi_1 \dots \psi_{m-1} \tilde{\psi}_m \quad \text{mit} \quad \tilde{\psi}_m := \psi_m C$$

eine Präfixdarstellung der Länge m von $\psi_1 \dots \psi_m C$. Daraus folgt die Behauptung durch Anwendung der Induktionsvoraussetzung.

q.e.d.

Definition 2.6

Sei $G = (N, \Sigma, P, S)$ eine EPG, $\gamma \in \Gamma^*$ sei ein BU-Präfix von G , $\psi_0 \psi_1 \dots \psi_k$ sei eine Präfixdarstellung von $\# \gamma$, dann gilt:

Für $i = 0, \dots, k$ ist die Menge $\text{Ges}(\psi_0 \dots \psi_i)$ der für $\psi_0 \dots \psi_i$ gesuchten Variablen definiert durch:

1. $\text{Ges}(\psi_0) = \text{Ges}(\#) := \{S\}$
2. $\text{Ges}(\psi_0 \dots \psi_i) := \{A \in N \mid \exists \# S \# \xrightarrow{R^*} \psi_0 \dots \psi_{i-1} A z \# \rightarrow \psi_0 \dots \psi_i \lambda_i z \#,$
 $z \in \Sigma^*, \lambda_i \in \Gamma^*\} \quad \text{für } i = 1, \dots, k$

Aus Satz 2.5 folgt, daß die Mengen von gesuchten Variablen stets von der leeren Menge verschieden sind.

Jede aus einem BU-Präfix mit Hilfe von Rechtsableitungen gewonnenen Zeichenfolge $u \in \Sigma^*$ ist ein Präfix eines Wortes der zugrunde liegenden Sprache. Deshalb ist der folgende Satz Grundlage für die Konstruktion der Analysefunktion F_4 , welche einen Analysator A_{F_4} induziert, welcher die LP-Eigenschaft hat.

Satz 2.6

Sei $G = (N, \Sigma, P, S)$ eine EPG, $\gamma \in \Gamma^*$ ein BU-Präfix, $\psi_0 \psi_1 \dots \psi_k$ eine Präfixdarstellung von $\# \gamma$, $a \in \Sigma$, dann gilt:

1. Im Fall von $\text{End}(\psi_k) \doteq a$ gilt:

γa ist BU-Präfix $\nexists A \in \text{Ges}(\psi_0 \dots \psi_k)$ mit $A \rightarrow \psi_k a \lambda_k \in P, \lambda_k \in \Gamma^*$

2. Im Fall von $\text{End}(\psi_k) \nrightarrow a$ gilt:

γa ist BU-Präfix $\nexists A \in \text{Ges}(\psi_0 \dots \psi_k)$ mit
 $A \rightarrow \psi_k C \lambda_k \in P \wedge C \xrightarrow{*} a \lambda, \lambda_k, \lambda \in \Gamma^*, C \in N$

3. Im Fall von $\text{End}(\psi_k) \triangleright a$ gilt:

Sei $B \in N$ die eindeutig bestimmte Variable mit $B \rightarrow \psi_k \in P$, dann ist $\psi_0 \psi_1 \dots \psi_{k-1} B$ genau dann ein BU-Präfix, wenn $B \in \text{Ges}(\psi_0 \dots \psi_k)$ gilt.

Bevor wir diesen Satz beweisen, berechnen wir die Mengen von gesuchten Variablen, welche in den Beispielen 2.5 - 2.7 auftraten. Damit wollen wir die Behauptungen 1 - 3 anschaulich erläutern.

Beispiel 2.8: Erläuterung von Satz 2.6

1. Zu "a"

Sei u die im Beispiel 2.5 definierte Eingabezeichenfolge.

Die Analyse von u mit Hilfe von F3 ergibt den Kellerinhalt

$\# Ph \text{ begin } if \ id$

Die Zeichenfolge $Ph \text{ begin } if \ id$ ist ein BU-Präfix.

Die Berechnung der gesuchten Variablen ergibt:

$\text{Ges}(\#) = \{S\}$

$\text{Ges}(\#Ph) = \{S\}$

$\text{Ges}(\#Ph \text{ begin}) = \{C\}$

$\text{Ges}(\#Ph \text{ begin } if) = \{I\}$

$\text{Ges}(\#Ph \text{ begin } if \ id) = \{F\}$

Eine Möglichkeit der Berechnung dieser Mengen wird am Beispiel von $\text{Ges}(\psi_0 \psi_1 \psi_2 \psi_3 \psi_4) := \text{Ges}(\#Ph \text{ begin } if \ id) = \{F\}$ gezeigt.

Wegen $\psi_4 = id$ muß für $A \in \text{Ges}(\psi_1 \dots \psi_4)$ gelten: $\exists A \rightarrow id \ \lambda \in P_1, \lambda \in \Gamma_1^*$

In G1 ergibt es drei Produktionen bei denen id ein Präfix der rechten Seite ist.

$IL \rightarrow id : | \ id, \ IL$

$A \rightarrow id : G \ E \ | \ id[E] : G \ E$

$F \rightarrow id : (E) \ | \ id[E]$

D.h. die Variablen IL, A, F sind möglicherweise Elemente von $\text{Ges}(\psi_0 \dots \psi_4)$.

Da zwischen if und A keine Präzedenzrelation gilt, existiert (wegen Hilfssatz 1.2) keine Rechtsableitung:

$$\#S\# \xrightarrow{R^*} \#\psi_1 \dots \psi_2 \text{ if } A \text{ z} \#$$

Deshalb gilt $A \notin \text{Ges}(\psi_0 \dots \psi_4)$. Auf analoge Weise zeigt man:

$$IL \notin \text{Ges}(\psi_0 \dots \psi_4).$$

Für die Variable F existiert eine solche Rechtsableitung, nämlich zum Beispiel:

$$\#S\# \xrightarrow{R^*} \#Ph \text{ begin if } F \text{ then id := id end.}\#$$

Insgesamt ergibt sich: $\text{Ges}(\psi_0 \dots \psi_4) = \{F\}$

Durch Anwendung von Satz 2.6 folgt, daß die Zeichenfolge

program id; begin if id :

kein Präfix von L_1 ist, da für keine Variable $A \in \text{Ges}(\psi_0 \dots \psi_4)$

$$A \rightarrow id : \lambda \in P_1, \lambda \in \Gamma_1^*$$

gilt.

2. Zu " $\triangleleft$ " :

Sei u die in Beispiel 2.6 definierte Zeichenfolge.

Die Analyse von u mit Hilfe von F3 ergibt den Kellerinhalt

#Ph var id :

Die Zeichenfolge *Ph var id :* ist ein BU-Präfix.

Analog früher zeigt man

$$\text{Ges}(\#Ph \text{ var id :}) = \{IL\}$$

Zwischen $:$ und $=$ gilt die Relation $\triangleleft$, deshalb ist die Zeichenfolge

program id ; var id :=

nach Satz 2.6 genau ein Präfix von L_1 , wenn eine Produktion $A \rightarrow id C \lambda$ in P_1 existiert mit:

$$1. C \xrightarrow{R^*} = \gamma, \gamma \in \Gamma_1^*$$

$$2. A \in \text{Ges}(\#Ph \text{ var id :})$$

Die Produktion

$$A \rightarrow id : G E$$

ist die einzige Produktion, welche die Forderung 1 erfüllt. Wegen

$A \in \text{Ges}(*Ph \text{ var } id :)$ folgt aus Satz 2.6, daß die Zeichenfolge

program id; var id :=

kein Präfix von L1 ist.

3. Der Fall " $\triangleright$ "

Sei u die in Beispiel 2.7 definierte Zeichenfolge

abfe

Die Analyse von u mit Hilfe von F3 ergibt den Kellerinhalt

abfe

Die Zeichenfolge *abf* ist ein BU-Präfix.

Wegen der Gültigkeit der Relation $\triangleright$ zwischen *f* und *e* ist Fall 3 von Satz 2.6 anwendbar, das heißt die Reduktion von *f* auf *E* liefert ein Präfix einer Rechtssatzform, wenn $E \in \text{Ges}(*abf)$ gilt. Wegen $\text{Ges}(*abf) = \{L\}$ ist *abfe* keine rechtskanonische Satzform über G2.

Δ

Alle in Beispiel 2.8 auftretenden Mengen von gesuchten Variablen waren einelementig. Das folgende Beispiel zeigt, daß dies nur zufällig so ist.

Beispiel 2.9: Eine Menge von gesuchten Variablen mit mehr als einem Element

Sei die Grammatik G2 aus Beispiel 2.7 gegeben. Dann gilt:

$$\begin{aligned} \text{Ges}(*a) &= \{A \in N2 \mid \exists *S* \xRightarrow{R,*} **A \lambda_0 * \wedge A \rightarrow a\lambda_1, \lambda_0, \lambda_1 \in \Gamma 2^*\} \\ &= \{A, H\} \end{aligned}$$

Δ

Der Beweis von Satz 2.6 ist eine unmittelbare Konsequenz aus der Definition der gesuchten Variablen.

Beweis von Satz 2.6

Beweis von (1)

" } "

Gelte $\text{End}(\gamma) \doteq a$ und sei γa ein BU-Präfix, dann ist zu zeigen:

$\exists A \in \text{Ges}(\psi_0 \dots \psi_k)$ mit $A \rightarrow \psi_k a \lambda_k \in P$, $\lambda_k \in \Gamma^*$.

Trivialerweise ist aber $\psi_0 \dots \psi_{k-1} \tilde{\psi}_k$, wo $\tilde{\psi}_k := \psi_k a$ eine Präfixdarstellung von $\# \gamma a$, d.h. es existiert ein $A \in \text{Ges}(\psi_0 \dots \psi_{k-1} \tilde{\psi}_k)$ mit:

$A \rightarrow \tilde{\psi}_k \lambda_k \in P$, daraus folgt, daß

$$A \in \text{Ges}(\psi_0 \dots \psi_{k-1} \psi_k)$$

gilt.

" < "

Triviale Folgerung der Definition der Mengen von gesuchten Variablen.

q.e.d. (1)

Beweis von (2)

" } "

Gelte $\text{End}(\gamma) \triangleleft a$ und sei γa ein BU-Präfix, dann ist zu zeigen:

$\exists A \in \text{Ges}(\psi_0 \dots \psi_k)$ mit $A \rightarrow \psi_k C \lambda_k \in P \wedge C \xRightarrow{*} a \lambda$, $\lambda_k, \lambda \in \Gamma^*$, $C \in N$.

γa ist BU-Präfix } "

$\psi_0 \dots \psi_k \psi_{k+1}$, mit $\psi_{k+1} = a$ ist Präfixdarstellung von $\# \gamma a$ mit $\text{End}(\# \gamma) \triangleleft a$ }

$\exists C \in N$ mit den Eigenschaften:

1. $C \xRightarrow{R,*} a \lambda$, $\lambda \in \Gamma^*$ und
2. $\text{End}(\# \gamma) \doteq C$ und
3. $\psi_1 \dots \psi_k C$ ist ein BU-Präfix

(Schlußweise analog Beweis von Satz 2.5).

} $\exists A \in \text{Ges}(\psi_0 \dots \psi_k C)$ mit $A \rightarrow \psi_k C \lambda \in P$.

" < "

Triviale Folgerung der Definition der Mengen von gesuchten Variablen.

q.e.d. (2)

Beweis von (3)

">"

Gelte $\text{End}(\gamma) > a$ und sei $\psi_1 \dots \psi_{k-1} B$ ein BU-Präfix, wo $B \rightarrow \psi_k \in P$, dann ist zu $B \in \text{Ges}(\psi_0 \dots \psi_k)$ zu zeigen.

Nach Voraussetzung gilt:

$$\exists S \xRightarrow{R^*} \psi_1 \dots \psi_{k-1} B \xRightarrow{R^*} \psi_1 \dots \psi_{k-1} \psi_k W$$

daus folgt nach der Definition der gesuchten Variablen $B \in \text{Ges}(\psi_0 \dots \psi_k)$.

"<"

trivial wie früher.

q.e.d. (3)

q.e.d. (Satz 2.6)

Um Satz 2.6 anwenden zu können, müssen die Mengen von gesuchten Variablen berechnet werden. Eine Möglichkeit dies auf effiziente Weise zu tun, bietet der folgende Satz.

Satz 2.7

Sei $G = (N, \Sigma, P, S)$ eine EPG. $\gamma \in \Gamma^*$ ein BU-Präfix von G , $\psi_0 \dots \psi_k$ eine Prefixdarstellung von $\# \gamma$, $a \in \Sigma$, dann gilt:

1. Gilt $\text{End}(\# \gamma) = a$ und ist γa ein BU-Präfix von G , dann folgt:

$$\text{Ges}(\# \gamma a) = \{A \in N \mid A \in \text{Ges}(\psi_0 \dots \psi_k) \wedge \exists A \rightarrow \psi_k a \lambda_k \in P, \lambda_k \in \Gamma^*\}$$

2. Gilt $\text{End}(\# \gamma) < a$ und ist γa ein BU-Präfix von G , dann folgt:

$$\text{Ges}(\# \gamma a) = \{A \in N \mid \exists B \in \text{Ges}(\psi_0 \dots \psi_k) \text{ mit:}$$

$$\exists B \rightarrow \psi_k \bar{C} \lambda_k \in P, \lambda_k \in \Gamma^* \wedge$$

$$\bar{C} \xRightarrow{R^*} A \bar{\lambda}, \lambda \in \Gamma^* \wedge$$

$$A \rightarrow a \lambda \in P, \lambda \in \Gamma^* \}$$

3. Gilt $\text{End}(\gamma) \triangleright a$ und $B \rightarrow \psi_k \in P$, wo $\psi_0 \dots \psi_{k-1} B$ ein BU-Präfix ist, dann folgt:

3.1 Falls $\text{End}(\psi_{k-1}) \doteq B$

$$\text{Ges}(\psi_0 \dots \psi_{k-1} B) = \{A \in N \mid A \in \text{Ges}(\psi_0 \dots \psi_{k-1}) \wedge A \rightarrow \psi_{k-1} B \lambda_{k-1} \in P, \lambda_{k-1} \in \Gamma^* \}$$

3.2 Falls $\text{End}(\psi_{k-1}) \triangleleft B$

$$\begin{aligned} \text{Ges}(\psi_0 \dots \psi_{k-1} B) &= \{A \in N \mid \exists C \in \text{Ges}(\psi_0 \dots \psi_{k-1}) \text{ mit:} \\ &\quad \exists C \rightarrow \psi_{k-1} \bar{C} \lambda_{k-1} \in P, \lambda_{k-1} \in \Gamma^* \wedge \\ &\quad \bar{C} \xrightarrow{R^*} A \bar{\lambda}, \bar{\lambda} \in \Gamma^* \wedge \\ &\quad A \rightarrow B \lambda \in P, \lambda \in \Gamma^* \} \end{aligned}$$

Beweis

Beweis von (1)

Sei $\widetilde{\text{Ges}}(\# \gamma a) := \{A \in N \mid A \in \text{Ges}(\psi_0 \dots \psi_k) \wedge A \rightarrow \psi_k a \lambda_k \in P, \lambda_k \in \Gamma^* \}$,

dann ist $\text{Ges}(\# \gamma a) = \widetilde{\text{Ges}}(\# \gamma a)$ zu zeigen.

Sei $A \in \text{Ges}(\# \gamma a) = \text{Ges}(\psi_0 \dots \psi_k a) \}$

$$\exists \# S \# \xrightarrow{R^*} \psi_0 \dots \psi_{k-1} A w \# \Rightarrow \psi_0 \dots \psi_{k-1} \psi_k a \lambda_k w \# , w \in \Sigma^* \}$$

$$\psi_0 \dots \psi_k \text{ ist BU-Präfix} \wedge A \in \text{Ges}(\psi_0 \dots \psi_k) \}$$

$$A \in \widetilde{\text{Ges}}(\# \gamma a) \}$$

$$\text{Ges}(\# \gamma a) \subseteq \widetilde{\text{Ges}}(\# \gamma a)$$

Sei $A \in \widetilde{\text{Ges}}(\# \gamma a)$

$$\exists A \in \text{Ges}(\psi_0 \dots \psi_k) \wedge A \rightarrow \psi_k a \lambda_k \in P \}$$

$$\exists \# S \# \Rightarrow \psi_0 \psi_1 \dots \psi_{k-1} A w \# \psi_0 \psi_1 \dots \psi_{k-1} \psi_k a \lambda_k w \# \}$$

$$A \in \text{Ges}(\psi_0 \dots \psi_k a) \}$$

$$\widetilde{\text{Ges}}(\# \gamma a) \subseteq \text{Ges}(\# \gamma a)$$

q.e.d. (1)

Beweis von (2)

Sei $\text{Ges}(\# \gamma a) := \{A \in N \mid \exists B \in \text{Ges}(\psi_0 \dots \psi_k) \wedge B \rightarrow \psi_k \bar{C} \lambda_k \in P, \lambda_k \in \Gamma^* \wedge$
 $\bar{C} \xrightarrow{*} A \bar{\lambda}, \bar{\lambda} \in \Gamma^* \wedge A \rightarrow a \lambda \in P, \lambda \in \Gamma^* \}$

Wieder ist $\text{Ges}(\# \gamma a) = \widetilde{\text{Ges}}(\# \gamma a)$ zu zeigen.

Sei $A \in \text{Ges}(\# \gamma a) = \text{Ges}(\psi_0 \dots \psi_k a)$

$$\exists \# S \# \xrightarrow{*} \psi_0 \dots \psi_k A w \# \Rightarrow \psi_0 \dots \psi_k a \lambda w \#, \lambda \in \Gamma^* \quad \}$$

$$\exists \bar{C} \in N \text{ mit } \text{End}(\psi_k) \doteq C, \exists \bar{C} \xrightarrow{*} A \lambda, \lambda \in \Gamma^* \quad \}$$

$$\exists \# S \# \xrightarrow{*} \psi_0 \dots \psi_k \bar{C} \bar{w} \# \quad \}$$

$$\exists B \in \text{Ges}(\psi_0 \dots \psi_k) \text{ mit } B \rightarrow \psi_k \bar{C} \lambda_k \in P \quad \}$$

$$A \in \text{Ges}(\# \gamma a) \quad \}$$

$$\text{Ges}(\# \gamma a) \subseteq \widetilde{\text{Ges}}(\# \gamma a)$$

Sei $A \in \widetilde{\text{Ges}}(\# \gamma a)$

$$\exists \# S \# \xrightarrow{*} \psi_0 \dots \psi_{k-1} B \bar{w} \# \Rightarrow \psi_0 \dots \psi_{k-1} \psi_k \bar{C} \lambda_k \bar{w} \# \xrightarrow{*} \psi_0 \dots \psi_k A \bar{\lambda} \bar{\lambda}_k \bar{w} \#$$

$$\xrightarrow{*} \psi_0 \dots \psi_k A w \# \Rightarrow \psi_0 \dots \psi_k a \lambda w \#, w \in \Sigma^* \quad \}$$

$$A \in \text{Ges}(\# \gamma a) \quad \}$$

$$\widetilde{\text{Ges}}(\# \gamma a) \subseteq \text{Ges}(\# \gamma a)$$

q.e.d. (2)

Der Beweis von 3 verläuft analog den Beweisen von 1 und 2.

q.e.d. (Satz 2.7)

Mit Hilfe des eben bewiesenen Satzes können die Mengen von gesuchten Variablen auf einfache Weise parallel zu den Schritten des Analysators A_{P3} bestimmt werden.

Im Fall der Gültigkeit von $\text{End}(\# \gamma) \doteq a$ ist die Menge $\text{Ges}(\# \gamma a)$ diejenige Teilmenge von $\text{Ges}(\# \gamma)$, für deren Elemente A gilt:

$$\exists A \rightarrow \psi_k a \lambda \in P, \lambda \in \Gamma^*$$

Bei entsprechender Darstellung aller Produktionen der Grammatik in der Form von verketteten Listen ist deshalb die Bestimmung von $\text{Ges}(\# \gamma a)$ in diesem Fall ein Teilproblem der Prüfung, ob eine Zeichenfolge rechte Seite einer Produktion ist. Also besteht der Mehraufwand, welchen die Bestimmung von $\text{Ges}(\# \gamma a)$ im Fall von $\text{End}(\# \gamma) \hat{=} a$ für den Analysator A_{F3} bedeutet in dem zusätzlichen Speicherplatz, der zur Darstellung der Produktionen in der Form von verketteten Listen benötigt wird.

Im Fall der Gültigkeit von $\text{End}(\# \gamma) \nrightarrow a$ erhält man die Menge $\text{Ges}(\# \gamma a)$ in zwei Schritten. Zunächst bestimmt man alle Variablen A zu denen eine Produktion der Gestalt $A \rightarrow a\lambda$, $\lambda \in \Gamma^*$ existiert. Die Menge $\text{Ges}(\# \gamma a)$ besteht dann aus allen Variablen $B \in \text{Ges}(\# \gamma) = \text{Ges}(\# \psi_1 \dots \psi_k)$ für welche

$$\exists B \rightarrow \psi_k \bar{C} \lambda_k \in P, \quad \lambda_k \in \Gamma^*, \bar{C} \in N$$

und

$$\bar{C} \xRightarrow{R}^* A \bar{\lambda}, \quad \lambda \in \Gamma^*$$

gilt. Die Menge aller Variablen A , für die eine Produktion $A \rightarrow a\lambda$, sowie die Menge aller Variablen $\bar{C}$ für die ein $B \in \text{Ges}(\# \gamma)$ mit $B \rightarrow \psi_k \bar{C} \lambda_k$ in P existiert, können mit Hilfe der oben genannten Listen ohne zusätzliche Suchschritte bestimmt werden. Die Prüfung ob zwischen einem Variablenpaar $\bar{C}, A$ eine Rechtsableitung der Form $\bar{C} \xRightarrow{R}^* A \bar{\lambda}$, $\bar{\lambda} \in \Gamma^*$ existiert, ist ein Teilproblem der Bestimmung der Anfangsmengen der Variablen einer Grammatik. Diese können bekanntlich im voraus berechnet werden. D.h. der zusätzliche Laufzeitaufwand für die Berechnung der Menge $\text{Ges}(\# \gamma a)$ im Fall von $\text{End}(\# \gamma) \nrightarrow a$ ist zurückgeführt auf die Speicherung der Relation $A \text{ Anf } B$ für alle Variablen der Grammatik und auf deren Abfrage für alle Paare $A, \bar{C}$ mit den oben genannten Eigenschaften.

Im Fall der Gültigkeit von $\text{End}(\# \gamma) \rightarrow a$ tritt, falls eine Variable $B \in \text{Ges}(\# \psi_1 \dots \psi_{k-1})$ mit $B \rightarrow \psi_k \bar{C} \lambda_k \in P$ existiert, einer der Fälle

$$\text{End}(\# \psi_1 \dots \psi_{k-1}) \hat{=} B \text{ oder } \text{End}(\# \psi_1 \dots \psi_{k-1}) \nrightarrow B$$

ein. Die zu berechnende Menge $\text{Ges}(\# \psi_1 \dots \psi_{k-1} B)$ gewinnt man nach dem oben beschriebenen Schema. D.h. der durch den Fall $\text{End}(\# \gamma) \rightarrow a$ zusätzlich hervorgerufene Laufzeitaufwand besteht in der Kellierung der Mengen von gesuchten Variablen nach jedem Berechnungsschritt.

Beispiel 2.10: Zur Berechnung der Mengen von gesuchten Variablen

Sei die L1-Zeichenfolge

```

program id ;
var id : id ;
begin
  :

```

gegeben. Die Zeichenfolge $\# \text{ program id ;}$ sei reduziert auf $\# Ph$.
Es gilt $\text{Ges}(\# Ph) = \{S\}$. Im folgenden berechnen wir mit Hilfe von Satz 2.7 die Mengen von gesuchten Variablen parallel zur Analyse:

1. Wegen $Ph \Leftarrow var$ gilt:

$$\begin{aligned} \text{Ges}(\# Ph \text{ var}) &= \{A \in N1 \mid \exists S \rightarrow Ph \ B \wedge B \xrightarrow{R}^* A \ \bar{\lambda}, A \rightarrow var \ \lambda \in P1\} \\ &= \{Vh\} \end{aligned}$$

2. Wegen $var \Leftarrow id$ gilt:

$$\begin{aligned} \text{Ges}(\# Ph \text{ var id}) &= \{A \in N1 \mid \exists Vh \rightarrow var \ IL1 \ id \wedge IL1 \xrightarrow{R}^* A \ \lambda \wedge A \rightarrow id \ \lambda \in P1\} \\ &= \{IL\} \end{aligned}$$

3. Wegen $id \dot{=} :$ gilt:

$$\begin{aligned} \text{Ges}(\# Ph \text{ var id :}) &= \{A \in N1 \mid \exists A \in \text{Ges}(\# Ph \text{ var id}) \wedge A \rightarrow id : \ \lambda \in P1\} \\ &= \{IL\} \end{aligned}$$

4. Wegen $: \triangleright id \wedge IL \rightarrow id : \in P1 \wedge var \Leftarrow IL$ gilt:

$$\begin{aligned} \text{Ges}(\# Ph \text{ var IL}) &= \{A \in N1 \mid \exists Vh \xrightarrow{R}^* var \ \bar{C} \wedge \bar{C} \xrightarrow{R}^* A \ \bar{\lambda} \wedge A \rightarrow IL \ \lambda \in P1\} \\ &= \{IL1\} \end{aligned}$$

5. Wegen $IL \triangleright id \wedge IL1 \rightarrow IL \in P1 \wedge var \dot{=} IL1$ gilt:

$$\begin{aligned} \text{Ges}(\# Ph \text{ var IL1}) &= \{A \in \text{Ges}(\# Ph \text{ var}) \mid \exists A \rightarrow var \ IL1 \ \lambda \in P1\} \\ &= \{Vh\} \end{aligned}$$

usw.

Δ

Im nächsten Abschnitt werden die Mengen von gesuchten Variablen zur Definition einer Analysefunktion F_4 verwendet, welche einen Analysator induziert, die die LP-Eigenschaft hat.

2.3.3 Ein Analyser, welcher Syntaxfehler frühestmöglich erkennt.

Aus Satz 2.6 folgt unmittelbar, daß eine Analysefunktion, welche vor jeder Veränderung des Kellers die Bedingungen 1 - 3 des Satzes 2.6 prüft, einen Analyser induziert, welcher die LP-Eigenschaft hat.

Satz 2.8

Sei $G = (N, Z, P, S)$ eine EPG, $L = L(G)$, Konf_G die Menge der Konfigurationen über G , dann gilt:

Die Abbildung $F4 : \text{Konf} \rightarrow \text{Konf}_G$, definiert durch:

$$F4(\# \alpha, u \#) := \begin{cases} (\# \alpha a, u') & \begin{array}{l} \text{falls} \\ 1. u \# = a u', a \in \Sigma, u' \in \Sigma^* \# \wedge \text{End}(\alpha) \neq a \\ 2. \exists A \in \text{Ges}(\psi_0 \dots \psi_k) \text{ mit:} \\ \quad A \rightarrow \psi_k a \lambda_k \in P, \lambda_k \in \Gamma^* \end{array} & (1) \\ (\# \alpha a, u') & \begin{array}{l} \text{falls} \\ 1. u \# = a u', a \in \Sigma, u' \in \Sigma^* \# \wedge \text{End}(\alpha) \neq a \\ 2. \exists B \in \text{Ges}(\psi_0 \dots \psi_k) \text{ mit:} \\ \quad B \rightarrow \psi_k C \lambda_k \in P \wedge C \xrightarrow{R} a \lambda, \lambda, \lambda_k \in \Gamma^* \end{array} & (2) \\ (\# \psi_1 \dots \psi_{k-1} B, u \#) & \begin{array}{l} \text{falls} \\ 1. \text{End}(\# \psi_1 \dots \psi_k) \neq \text{Anf}(u \#) \\ 2. \exists B \in \text{Ges}(\psi_0 \dots \psi_k) \text{ mit } B \rightarrow \psi_k \in P \end{array} & (3) \\ \text{true} & \begin{array}{l} \text{falls} \\ 1. k = 1 \wedge \psi_1 = S \wedge u = \varepsilon \end{array} & (4) \\ |u \#| - 1 & \text{sonst} & (5) \end{cases}$$

wo α ein BU-Präfix, $\psi_0 \dots \psi_k$ eine Präfixdarstellung von $\# \alpha$ ist, ist eine Analysefunktion für L mit der Eigenschaft:

Sei $u \in Z^+$ mit $(\# , u \#) F4^+ (\# \alpha, u \#)$, dann ist α ein BU-Präfix von G .

Beweis:

F4 unterscheidet sich von F3 nur im Zeitpunkt der Fehlererkennung, d.h. F4 ist eine Analysefunktion. Aus Satz 2.6 folgt, daß die Kellerinhalte von Konfigurationen, welche durch schrittweise Anwendung von F4 auf eine Anfangskonfiguration erzeugt wurden, genau dann BU-Präfixe sind, wenn die Konfigurationen von den Endekonfigurationen verschieden sind.

q.e.d.

D.h. durch Satz 2.8 ist das Problem der Konstruktion eines Analysators für einfache Präzedenzsprachen mit LP-Eigenschaft theoretisch gelöst. Satz 2.7 zeigt, daß die in den Positionen (1) bis (3) der Definition von F4 durchzuführenden Prüfungen ein Teilproblem der Berechnung der Mengen von gesuchten Variablen sind. Deshalb ist der zusätzliche Bedarf an Rechenzeit und Speicherplatz, den eine Implementierung von A_{F4} gegenüber einer solchen von A_{F3} zur Laufzeit hat, gegeben durch den Aufwand für die Berechnung und Kellerung der Mengen von gesuchten Variablen. Wie in Anschluß an Satz 2.7 erläutert wurde, besteht bei entsprechender Implementierung der Aufwand für die Berechnung im wesentlichen in der Speicherung der Relation A Anf B für alle Paare von Variablen der zu Grunde liegenden Grammatik, sowie deren Abfrage für kleine Mengen (siehe unten) von Variablen. Der Aufwand für die Kellerung der Mengen von gesuchten Variablen ist unbedeutend, da die Anzahl der Elemente einer Menge von gesuchten Variablen begrenzt ist durch die maximale Anzahl von Variablen zu denen Produktionen existieren, deren rechte Seiten mit demselben Element beginnen.

Implementiert man den Analysator A_{F4} mit Hilfe der von F.Hauger [Hau.77] vorgeschlagenen einfachen Präzedenzgrammatik für Pascal [Wir,Jen.74], so ergeben sich für die Anzahl der Elemente der, nach obigen Ausführungen, zu speichernden Mengen die folgenden Werte:

1. Die Grammatik enthält 233 Variable, also kann die Relation A Anf B mit Hilfe von 4 mal 233 60-Bit Worten dargestellt werden.
2. Die maximale (durchschnittliche) Anzahl von Variablen, zu denen

Produktionen existieren, deren rechte Seite mit demselben Symbol beginnt, ist 5 (2,2). D.h. daß die Mengen von gesuchten Variablen höchstens fünf, im Durchschnitt aber nur 2,2 Elemente enthalten.

Aus diesen Überlegungen kann nach üblichen programmiertechnischen Erfahrungen gefolgert werden, daß der zusätzliche Laufzeitaufwand, den eine Implementierung von A_{F4} gegenüber einer solchen von A_{F3} für gängige höhere Programmiersprachen benötigt, keine wesentliche Verschlechterung der Effizienz der Syntaxanalyse bedeutet.

Im nächsten Abschnitt werden ausgehend von der Diskussion der bisher bekannten Verfahren eine Klasse von neuen Methoden der Behandlung von Syntaxfehlern bei einfachen Präzedenzsprachen entwickeln. Es wird sich herausstellen, daß die Qualität dieser Methoden davon abhängt, wie früh die zu Grunde liegende Analysealgorithmen Fehler erkennen. Durch eine Erweiterung des Begriffs der Mengen von gesuchten Variablen werden wir einen optimalen Fehlerbehandlungsalgorithmus erhalten.

3. NEUE METHODEN DER BEHANDLUNG VON SYNTAXFEHLERN BEI EINFACHEN PRÄZEDENZSPRACHEN

In diesem Abschnitt werden, ausgehend von dem Versuch, ein zeichengesteuertes Wiederaufsetzverfahren zu konstruieren, welches mit jedem Zeichen die Analyse fortsetzt, neue "fehlerbehandelnde" Analysealgorithmen gewonnen. Von ihnen wird behauptet, daß sie den bisher bekannten Algorithmen überlegen sind. Um diese Behauptung begründen zu können, wird zunächst ein Fehlerbegriff definiert, mit Hilfe dessen die Qualität von Fehlerbehandlungsalgorithmen beurteilt werden kann. Dann werden die drei bekanntesten Verfahren zur Behandlung von Syntaxfehlern bei einfachen Präzedenzsprachen grob dargestellt, um sie mit den im Anschluß daran entwickelten Verfahren vergleichen zu können.

3.1 Der Fehlerbegriff als Grundlage für die Diskussion von fehlerbehandelnden Analysealgorithmen

Wie bereits in der Einleitung erwähnt, ist das Hauptproblem jeder Fehlerbehandlung die Tatsache, daß dem fehlerbehandelnden Algorithmus nur die fehlerhafte, nicht aber die vom Programmierer beabsichtigte Zeichenfolge bekannt ist. Jede Fehlerdiagnose beruht deshalb auf einer Annahme über die Art des tatsächlichen Fehlers. Ist L die zu analysierende Sprache, Σ das Alphabet von L , dann ist durch die Menge aller dieser Annahmen eine Abbildung von Σ^* nach $2^L \setminus \{\emptyset\}$ definiert. Jede solche Abbildung bezeichnen wir als einen Fehlerbegriff.

Definition 3.1

Sei L formale Sprache über dem Alphabet Σ , dann gilt:

Jede Abbildung $F : \Sigma^* \rightarrow 2^L \setminus \{\emptyset\}$ mit $F(x) = \{x\}$ für $x \in L$ heißt ein Fehlerbegriff für L .

Damit ein Fehlerbegriff brauchbar ist, muß man von ihm verlangen, daß er in möglichst vielen Fällen die fehlerhafte auf die vom Programmierer beabsichtigte Zeichenfolge abbildet.

Von den bisher bekannten Fehlerbegriffen scheint der im folgenden, mit Hilfe des Hammingabstandes definierte diese Forderung am besten zu erfüllen.

Definition 3.2

Sei Σ ein Alphabet, L eine formale Sprache über Σ , $x \in \Sigma^*$, dann gilt:

1. Die Abbildungen I , D , R von Σ^* nach $2^{\Sigma^*} \setminus \{\emptyset\}$ sind definiert durch:

1. $I(x) := \{y \in \Sigma^* \mid \exists u, v \in \Sigma^* \wedge a \in \Sigma \text{ mit: } x = uv \wedge y = uav\}$
2. $D(x) := \{y \in \Sigma^* \mid \exists u, v \in \Sigma^* \wedge a \in \Sigma \text{ mit: } x = uav \wedge y = uv\}$
3. $R(x) := \{y \in \Sigma^* \mid \exists u, v \in \Sigma^* \wedge a, b \in \Sigma \text{ mit: } x = uav \wedge y = ubv\}$

2. Die Abbildungen I (Insertion), D (Deletion), R (Replacement) werden als (elementare) Korrekturoperatoren bezeichnet.

3. Unter einer Korrektur der Zeichenfolge x verstehen wir die Anwendung eines Korrekturoperators.

4. Die Menge der Zeichenfolgen, die mit Hilfe einer Korrektur aus x erhalten werden können, bezeichnen wir mit $C(x)$. Sie ist definiert durch:

$$C(x) := I(x) \cup D(x) \cup R(x).$$

5. Die Menge der Zeichenfolgen, die mit Hilfe einer Korrektur auf ein Wort von L abgebildet werden können, bezeichnen wir mit $C(L)$. Sie ist definiert durch:

$$C(L) := \{x \in \Sigma^* \mid C(x) \cap L \neq \emptyset\}$$

6. Die Menge der Zeichenfolgen, die durch maximal n ($n \in \mathbb{N}^0$) Korrekturen aus x erhalten werden können, bezeichnen wir mit $C^n(x)$. Wir definieren sie mit Hilfe vollständiger Induktion durch:

$$C^0(x) := \{x\}$$

$$C^n(x) := \{y \in \Sigma^* \mid \exists z \in C^{n-1}(x) : y \in C(z)\} \quad \text{für } n > 1$$

7. Analog bezeichnen wir die Menge der Wörter, die mit Hilfe von maximal n ($n \in \mathbb{N}^0$) Korrekturen auf L abgebildet werden können mit $C^n(L)$. Sie ist definiert durch:

$$C^n(L) := \{y \in \Sigma^* \mid \exists x \in L : y \in C^n(x)\}.$$

Unter dem Hammingabstand von x, y verstehen wir die minimale Anzahl von Korrekturen, die nötig ist, um x nach y abzubilden.

Definition 3.3

Sei Σ ein Alphabet, $x, y \in \Sigma^*$, $i \in \mathbb{N}^0$, dann gilt:

Der Hammingabstand $HA(x, y)$ ist definiert durch:

$$HA(x, y) = i : \nexists \exists x \in C^i(y) \wedge x \notin C^j(y) \text{ für } 0 \leq j < i$$

Mit Hilfe des Hammingabstandes definieren wir den für diese Arbeit gültigen Fehlerbegriff Ham_L .

Definition 3.4

Sei L formale Sprache über dem Alphabet Σ , $x \in \Sigma^*$, $i \in \mathbb{N}^0$, dann gilt:

1. Der Fehlerbegriff Ham_L ist definiert durch

$$\text{Ham}_L(x) := \{y \in L \mid HA(x, y) = \min_{z \in L} HA(x, z)\}$$

2. Jedes Element aus Ham_L heißt eine für x beabsichtigte Zeichenfolge.

3. x enthält i Fehler : $\nexists$

$$\exists y \in \text{Ham}_L(x) : HA(x, y) = i$$

Durch den obigen Fehlerbegriff ist nur die Anzahl der Fehler in einer Zeichenfolge eindeutig definiert. Die Menge der statt einer Zeichenfolge x beabsichtigten Zeichenfolgen ist im allgemeinen nicht einelementig (siehe Bsp. 3.1,1). Ort und Art der Fehler in einer Zeichenfolge kann

man durch Ort und Art der Korrekturen definieren, die notwendig sind, um die Zeichenfolge x in eine der beabsichtigten Zeichenfolgen abzubilden.

Definition 3.5

Sei Σ ein Alphabet, $x, y \in \Sigma^*$ mit $y \in C(x)$, dann gilt:

1. j heißt eine mögliche Korrekturstelle in x bzgl. y , wenn einer der folgenden Fälle zutrifft:

$$1.1 \quad \exists u, v \in \Sigma^* \wedge a \in \Sigma \text{ mit : } x = uv \wedge y = uav \wedge j = |x| - |v| + 1$$

$$1.2 \quad \exists u, v \in \Sigma^* \wedge a \in \Sigma \text{ mit : } x = uav \wedge y = uv \wedge j = |x| - |v|$$

$$1.3 \quad \exists u, v \in \Sigma^* \wedge a, b \in \Sigma \text{ mit : } x = uav \wedge y = ubv \wedge j = |x| - |v|$$

2. Das Maximum aller möglichen Korrekturstellen von x bzgl. y bezeichnen wir als die Korrekturstelle $KS(x, y)$.

Definition 3.6

Sei L formale Sprache über dem Alphabet Σ , $x \in \Sigma^*$ eine Zeichenfolge mit n Fehlern, $n \in \mathbb{N}$, $y \in \text{Ham}_L(x)$ und $x_0, \dots, x_n \in \Sigma^*$ mit

$$1. \quad x_0 = x$$

$$2. \quad x_i = C(x_{i-1}) \quad \text{für } i = 1, \dots, n$$

$$3. \quad KS(x_{i-1}, x_i) \leq KS(x_i, x_{i+1}) \quad \text{für } i = 1, \dots, n-1$$

$$4. \quad x_n = y$$

dann gilt für $x_q \in \{x_0, \dots, x_{n-1}\}$:

1. Das Paar (x_q, x_{q+1}) definiert den Fehler (D, a, j) in x :

$$\exists u, v \in \Sigma^* \wedge a \in \Sigma \wedge j \in \mathbb{N} \text{ mit : } x_q = uv \wedge x_{q+1} = uav \wedge j = |x| - |v| + 1$$

2. Das Paar (x_q, x_{q+1}) definiert den Fehler (I, a, j) in x :

$$\exists u, v \in \Sigma^* \wedge a \in \Sigma \wedge j \in \mathbb{N} \text{ mit: } x_q = uav \wedge x_{q+1} = uv \wedge j = |x| - |v|$$

3. Das Paar (x_q, x_{q+1}) definiert den Fehler $(R, (a, b), j)$ in x :

$$\exists u, v \in \Sigma^* \wedge a, b, c \in \Sigma \wedge j \in \mathbb{N} \text{ mit } x_q = uav \wedge x_{q+1} = ubv \wedge j = |x| - |v|$$

4. In den obigen drei Fällen heißt j eine (mögliche) Fehlerstelle in x

5. Jede Folge $x_0, \dots, x_n$ mit den Eigenschaften 1 bis 4 heißt eine Korrekturfolge für x . Jede durch eine Korrekturfolge $x_0, \dots, x_n$ definierte Folge von Fehlern in x heißt eine Fehlerdiagnose für x . Eine Fehlerdiagnose heißt kanonisch, wenn die Folge der durch sie definierten Fehlerstellen identisch mit der Folge der Korrekturstellen ist.

Bemerkung 3.1

Die Forderung 3 der Definition 3.6 ist stets durch Umordnen der x_j zu erreichen. Sie bewirkt, daß die der Korrekturstelle folgende Zeichenfolge immer ein Suffix von x ist, d.h. daß insbesondere jede Fehlerstelle j einer kanonischen Fehlerdiagnose ein Element der Zeichenfolge x bezeichnet. Zum Beispiel bedeutet bei einer kanonischen Fehlerdiagnose der Fehler (D, a, j) in x , daß vor dem j -ten Zeichen der Zeichenfolge x das Zeichen a weggelassen wurde. Der Fehler (I, a, j) bedeutet, daß das j -te Zeichen (a) falscherweise eingeschoben wurde.

Damit sind für eine gegebene Zeichenfolge x die Begriffe: für x beabsichtigte Zeichenfolge; Anzahl der Fehler in x ; Fehlerdiagnose für x definiert. Leider ist nur die Anzahl der Fehler in x eindeutig festgelegt, denn im allgemeinen ist weder die Menge der für x beabsichtigten Zeichenfolgen, noch die Menge der Fehlerdiagnosen einelementig.

Beispiel 3.1: Beabsichtigte Zeichenfolgen, Fehlerdiagnosen, Fehlerstellen

1. Beabsichtigte Zeichenfolgen

Die bezüglich L1 fehlerhafte Zeichenfolge x sei definiert durch

```
x := program id;  
      begin  
          id := id ++ id  
      end.
```

Sie enthält einen Fehler.

Die Menge der beabsichtigten Zeichenfolgen enthält zwei Elemente y_1, y_2 , nämlich

```
y1 := program id; begin id := id + id end.
```

und

```
y2 := program id; begin id := id + id + id end.
```

2. Mögliche Fehlerdiagnosen für eine beabsichtigte Zeichenfolge

Sei die bezüglich L1 fehlerhafte Zeichenfolge x definiert durch

```
x := program id;  
      begin  
          [ id id ] := id  
      end.
```

Sie enthält zwei Fehler.

Die eindeutig bestimmte beabsichtigte Zeichenfolge y ist

```
program id;  
begin  
    id [ id ] := id  
end.
```

Es gibt jedoch genau vier mögliche Fehlerdiagnosen für x .

1. x enthält die Fehler $(D, id, 5)$ und $(I, id, 6)$ ⁽¹⁾
2. x enthält die Fehler $(D, id, 5)$ und $(I, id, 7)$
3. x enthält die Fehler $(I, [, 5)$ und $(D, [, 7)$
4. x enthält die Fehler $(R, ([, id), 5)$ und $(R, (id, [), 6)$

Das heißt auch bei eindeutig bestimmter beabsichtigter Zeichenfolge ist die Fehlerdiagnose für x nicht eindeutig bestimmt.

3. Fehlerstellen in x

Sei $\Sigma = \{a, b, c, d, e\}$, $L = \{aaabcd\}$.

Die fehlerhafte Zeichenfolge x sei definiert durch $x := adc$

Die Folge x_0, x_1, x_2, x_3, x_4 , definiert durch:

$$x_0 = adc$$

$$x_1 = aadc$$

$$x_2 = aaade$$

$$x_3 = aaabc$$

$$x_4 = aaabcd$$

ist eine Korrekturfolge für x . Die dadurch definierte Fehlerdiagnose f_1, f_2, f_3, f_4 ist:

$$f_1 = (D, a, 2)$$

$$f_2 = (D, a, 2)$$

⁽¹⁾ Zwischenräume sind keine Teilzeichenfolgen von x , sie dienen nur einer übersichtlichen Notation.

$$\begin{aligned} f_3 &= (R, (\bar{d}, b), 2) \\ f_4 &= (D, e, 4) \end{aligned}$$

Δ

In dieser Arbeit wird die Beurteilung der Qualität eines Fehlerbehandlungsalgorithmus durch den Vergleich der von ihm erzeugten Fehlermeldungen mit den möglichen Fehlerdiagnosen erfolgen. Im idealen Fall sollte die Menge der Fehlermeldungen mit einer möglichen Fehlerdiagnose übereinstimmen. Dies wird jedoch in der Regel nur anhand von einigen Beispielen und auf heuristische Weise geschehen können, da die Fehlermeldungen die von Fehlerbehandlungsalgorithmen erzeugt werden, meist in anderer Form notiert sind, als die einer Fehlerdiagnose in unserem Sinn.

Um wenigstens die Anzahl der von einem Fehlerbehandlungsalgorithmus gemeldeten Fehler mit der Anzahl der Fehler in x vergleichen zu können, definieren wir die Begriffe fehlerbehandelnder Analysator (Analysealgorithmus) sowie Anzahl der gemeldeten Fehler eines fehlerbehandelnden Analysators.

Definition 3.7

Sei L formale Sprache über dem Alphabet Σ . FM eine Menge mit $FM \setminus \{\text{true}\} = \emptyset$, dann gilt:

1. Eine Abbildung $A : \Sigma^* \rightarrow \{\text{true}\} \cup FM^+$ heißt genau dann ein fehlerbehandelnder Analysator bzw. Analysealgorithmus (für L), wenn gilt:

$$A(x) = \begin{cases} \text{true} & \text{falls } x \in L \\ f \in FM^+ & \text{falls } x \notin L \end{cases} .$$

2. FM heißt die Menge der Fehlermeldungen von A .

3. j heißt die Anzahl der (von A) gemeldeten Fehler in x : 

$$A(x) = f \in FM^+ \wedge |f| = j$$

4. Sei i die Anzahl der Fehler in x und j die Anzahl der von A in x ge-

meldeten Fehler, wo $j > i$ gilt, dann gilt die Sprechweise: A verur-
sacht (meldet) $(j - i)$ Folgefehler.

Bemerkung 3.2

Die im Abschnitt 1 eingeführten Analysatoren sind fehlerbehandelnde Analysatoren, welche nur höchstens einen Fehler melden. Gemäß der in der Einleitung unter Punkt E1 geforderten Eigenschaft sollte ein fehlerbehandelnder Analysator für eine Zeichenfolge mit n Fehlern mindestens n Fehler melden, und diese sollten sinngemäß mit einer möglichen Fehlerdiagnose übereinstimmen. Im folgenden wird sich herausstellen, daß die bisher bekannten fehlerbehandelnden Analysatoren in der Regel Folgefehler melden.

3.2 Grundzüge und Diskussion der bisher bekannten Methoden

In der Literatur werden drei Methoden zur Behandlung von Syntaxfehlern bei einfachen Präzedenzsprachen beschrieben. Es sind dies die Verfahren von Wirth [Wir,68] , Leinius [Lei,70] und Rhodes [Rho,73] . Die drei Verfahren haben gemeinsam die Eigenschaft, daß sie nach dem Erkennen eines Fehlers in der Eingabezeichenfolge den Kellerinhalt, bzw. in manchen Fällen die noch zu analysierende Zeichenfolge, so zu verändern versuchen, daß die Fortsetzung der Analyse möglich wird.

Den Korrekturprozeß kann man sich dabei grob aus zwei Teilen zusammengesetzt denken:

1. Der Kandidatenwahl
2. Der Auswahl einer 'Korrektur' für den Kandidaten.

Unter der Kandidatenwahl verstehen wir die Ermittlung einer, im folgenden mit Kandidat bezeichneten Teilzeichenfolge, der durch Konkatenation von Kellerinhalt und dem noch zu analysierenden Suffix der Eingabezeichenfolge entstandenen Zeichenfolge.

Mögliche Korrekturen des Kandidaten sind die Streichung des Kandidaten oder die Ersetzung des Kandidaten durch ein nichtterminales Zeichen des Gesamtalphabets.

Beispiel 3.2: Kandidatenwahl und Korrektur

Die bezüglich L1 fehlerhafte Zeichenfolge sei

```
program id,  
begin  
    id := id [ ( id + id ] + id * id  
end.
```

Der Analysator erkenne die Existenz eines Fehlers beim Lesen der ersten schließenden Klammer.

Zu diesem Zeitpunkt ist der Kellerinhalt

$\# Pi \text{ begin id} : G id [(E$

Denkbare Kandidaten sind dann:

1. K1 =]
2. K2 = (E]
3. K3 = id [(E] + id * id

Denkbare Korrekturen sind:

1. Ersetzen von K1 durch)
2. Ersetzen von K2 durch eine der Variablen: F, T, T1, E
3. Ersetzen von K3 durch E

Die drei Methoden unterscheiden sich durch die Strategien mit Hilfe welcher die Kandidatenwahl bzw. die Auswahl der Korrektur durchgeführt wird. Deshalb wird in den folgenden drei Abschnitten versucht, jeweils die Grundideen herauszuarbeiten, auf denen Kandidatenwahl bzw. Korrektur beruhen. Auf Einzelheiten der Algorithmen wird dabei bewußt verzichtet.

3.2.1 Die Methode von Wirth

Die Methode von Wirth ist eingebettet in eine leicht modifizierte Form des von der Analysefunktion F1 (siehe Satz 1.2) induzierten Analysators. Die Modifikation besteht darin, daß zwischen dem linken Randzeichen und

jedem Zeichen des Alphabets die Relation $\ll$, sowie zwischen dem rechten Randzeichen und jedem Zeichen des Alphabets die Relation $\gg$ als gültig angesehen wird. Deswegen erkennt dieser Analysator Syntaxfehler in genau zwei Situationen, nämlich entweder, wenn beim Lesen von links nach rechts ein Zeichenpaarfehler entdeckt wird, oder wenn beim Versuch einer Reduktion ein Reduktionsfehler entdeckt wird.

Im Fall eines Zeichenpaarfehlers wird in einer vorgegebenen Liste IL, von terminalen Zeichen, nach einem Zeichen gesucht, welches mit dem obersten Kellerelement, sowie mit dem aktuellen Zeichen eine Präzedenzrelation hat. Das erste in der Liste IL gefundene Zeichen mit diesen Eigenschaften wird in die Eingabezeichenfolge vor dem aktuellen Zeichen eingeschoben. Dann wird die Analyse fortgesetzt. Existiert kein solches Zeichen, so wird das aktuelle Zeichen, welches die Fehlersituation verursachte, gekellert und die Analyse fortgesetzt. In diesem Fall wird stets im Verlauf der Analyse ein Reduktionsfehler auftreten.

Im zweiten Fall wird die den Reduktionsfehler auslösende Zeichenfolge mit einer vorgegebenen Liste PL von fehlerhaften Produktionen auf Identität verglichen. Ist die fehlerauslösende Zeichenfolge mit einer der fehlerhaften Produktionen identisch, so wird die dazugehörige linke Seite an Stelle der fehlerhaften Zeichenfolge gekellert und die Analyse fortgesetzt. Ist dies nicht der Fall, so wird die Analyse beendet. ⁽¹⁾

Bei dem Wirthschen Verfahren sind also Kandidatenwahl und Korrektur abhängig von der jeweiligen Fehlersituation.

Im Falle eines Zeichenpaarfehlers wird zunächst angenommen, daß der Fehler durch Auslassung eines Zeichens verursacht wurde. Deshalb ist der Kandidat die leere Zeichenfolge und die zugelassenen Korrekturen sind Einschiebungen von Zeichen. Ist eine solche Einschiebung nicht möglich, so wird das Zeichen gekellert. Dadurch wird die Behandlung des Fehlers verschoben auf einen späteren Zeitpunkt der Analyse. Dort wird er dann als Reduktionsfehler behandelt.

⁽¹⁾ Diese Aussage ist eine Vermutung, denn Wirth beschreibt die Behandlung dieses Falles in seiner Arbeit nicht.

Im Falle eines Reduktionsfehlers ist der Kandidat die zu reduzierende Zeichenfolge. Die Korrektur beruht auf der Vorstellung, daß der überwiegende Teil solcher Fehlersituationen bekannt ist und deshalb mit Hilfe vorgegebener Fehlerproduktionen behandelt werden kann.

Beispiel 3.3 : Die Methode von Wirth

Die bezüglich L1 fehlerhafte Zeichenfolge sei

```

program id;
var
    id, id : id
begin
    id := id [ ( id + id ] + id * id
end.

```

Zwischen *id* und *begin* besteht keine Präzedenzrelation, deshalb wird beim Verarbeiten des Zeichens *begin* ein Fehler entdeckt. Der Kellerinhalt dabei ist

* Ph var IL1 id

In der Liste IL wird jetzt nach einem Zeichen z gesucht für welches $id \odot z$ und $z \odot begin$ gilt.⁽¹⁾

Das Zeichen ; ist ein solches Zeichen. Wird es als erstes in IL gefunden, so wird es als aktuelles Zeichen aufgefaßt und die Analyse kann fortgesetzt werden. In diesem Fall wird der nächste Fehler nach dem Kellern von] entdeckt. Der Kellerinhalt ist jetzt

* Ph V begin id : G id [(E]

, das aktuelle Zeichen ist .

Wegen [< (und] > + wird eine Reduktion der Zeichenfolge (E] versucht, und dabei wird ein Reduktionsfehler entdeckt. Befindet sich in der Liste der fehlerhaften rechten Seiten die Zeichenfolge (E]

¹ Die Relation $\odot$ gilt genau dann zwischen zwei Zeichen X,Y, wenn eine der Relationen $X < Y$, $X = Y$, $X > Y$ gilt.

als rechte Seite für F , so wird für $(E]$ die Variable F gekellert. Diese Kellerkorrektur entspricht einer Ersetzung von $]$ durch $)$. In dem obigen Kontext ist dies keine Korrektur mit minimalem Abstand. Deshalb wird nach dem Lesen des letzten Zeichens id ein weiterer Reduktionsfehler erkannt. Der Kellerzustand dabei ist

$$Ph \ V \ begin \ id : G \ id \ [\ E$$

, das aktuelle Zeichen ist end

Jetzt wird versucht, die Zeichenfolge $id \ [\ E$ zu reduzieren. Befindet sich diese Zeichenfolge in der Liste PL als rechte Seite für F , so kann die Analyse ohne weitere Fehler fortgesetzt werden.

Δ

An obigem Beispiel erkennt man die Vor- bzw. Nachteile der Wirthschen Methode deutlich.

Die Vorteile sind:

1. Die Methode ist sehr einfach zu implementieren.
2. Die Analyse korrekter Programme wird durch die Existenz der Fehlerbehandlung nicht ineffizienter.
3. Sind die Listen IL und PL abgestimmt auf die am häufigsten auftretenden Fehlertypen, so kann die Fehlerbehandlung Ergebnisse liefern, welche für Anwendungen in der Praxis zumindest ausreichend sind.

Die Nachteile sind:

1. Die Methode ist bisher nicht automatisch erzeugbar, d.h. es existieren keine systematischen Aussagen zur Definition der Listen IL und PL in Abhängigkeit der zugrunde liegenden Grammatik.
2. Auch bei optimaler Wahl der Listen IL und PL werden von dem Verfahren zahlreiche Folgefehler produziert, da die Wahl der Korrektur im wesentlichen unabhängig vom Kontext des Fehlers geschieht (siehe 2. Fehler im Beispiel 3.3).
3. Auch bei systematischer Wahl der Listen IL und PL sind systematische Aussagen über Art und Anzahl der Folgefehler, d.h. über die Qualität

der Methode nur sehr eingeschränkt denkbar.

3.2.2 Die Methode von Leinius

Die Methode von Leinius ist in den von der Analysefunktion F_2 (siehe Satz 2.1) induzierten Analysator A_{F_2} eingebettet. Wie bereits in Abschnitt 2.2.2 erläutert, erkennt A_{F_2} die mit Zeichenpaarfehler, Reduktionsfehler und Kellerungsfehler bezeichneten Fehlertypen.

Sei zum Zeitpunkt der Fehlererkennung der Kellerinhalt durch $\psi_0 \psi_1 \dots \psi_k$ dargestellt. Für $\psi_0 \psi_1 \dots \psi_k$ gelten die Forderungen 1, 2 und 4 aus Satz 2.3. ψ_i repräsentiere das aktuelle Zeichen. Dann ist ψ_k der erste Kandidat für eine mögliche Korrektur des Kellers.

Unter einer Korrektur wird die Ersetzung des Kandidaten durch eine Variable verstanden. Eine Variable wird dabei genau dann als Ersetzung für den Kandidaten zugelassen, wenn sie die eindeutig bestimmte Variable mit der Eigenschaft ist, mit ihrem Links- und Rechtskontext verträglich zu sein. Das bedeutet, daß die Ersetzung des Kandidaten durch diese Variable nicht automatisch einen weiteren Fehler verursacht.

Die Bestimmung der zum Links- und Rechtskontext verträglichen Variablen geschieht in drei Schritten:

1. Bestimmung der Menge LK der zum Linkskontext verträglichen Variablen.
2. Bestimmung der Menge RK der zum Rechtskontext verträglichen Variablen.
3. Bestimmung der Menge LRK der zum Links- und Rechtskontext verträglichen Variablen (i.allg. gilt $LRK \subseteq LK \cap RK$ und $LRK \neq LK \cap RK$).

Um die Menge LK zu definieren, berechnet Leinius zunächst diejenige Zahl j , für die gilt:

$$j := \begin{cases} \max_{i=0, \dots, k} (|\psi_i| = 1) & \text{falls } \exists \psi_i : |\psi_i| = 1 \\ 1 & \text{sonst} \end{cases}$$

, wobei die Mengen V_i durch

$$V_i := \{A \in N \mid \exists A \rightarrow \psi_i \alpha_i \in P, \alpha_i \in \Gamma^*\}, i = 0, \dots, k$$

definiert sind.

Dann werden ausgehend von $\bar{V}_j := V_j$ für $i = j + 1, \dots, k$ die Mengen $\bar{V}_i$ nach der Vorschrift

$$\bar{V}_i := \{A \in V_i \mid \exists B \in \bar{V}_{i-1} \text{ mit } B \rightarrow \psi_{i-1} A \lambda_{i-1} \in P \wedge A \xrightarrow{R^*} \psi_i \lambda_i, \lambda_{i-1} \lambda_i \in \Gamma^*\}$$

berechnet.

Die Menge LK der zum Linkskontext verträglichen Variablen ist dann die Menge V_k .

Die Menge der zum Rechtskontext verträglichen Variablen ist definiert durch

$$RK := \{A \in N \mid A \odot I\}$$

Die Menge der zum Rechts- und Linkskontext verträglichen Variablen LRK ist die Menge der Variablen $A \in LK \cap RK$ für die eine der folgenden Bedingungen gilt:

1. Aus $\text{End}(\psi_k) \dot{=} A$ und $A \triangleright I$ folgt die Existenz einer Produktion $B \rightarrow \psi_{k-1} A$ in P
2. Aus $\text{End}(\psi_k) \dot{=} A$ und $A \dot{=} I$ folgt die Existenz einer Produktion $B \rightarrow \psi_{k-1} A \tau$ in P , $\tau \in \Gamma^*$
3. Aus $\text{End}(\psi_{k-1}) \dot{=} A$ und $A \triangleleft I$ folgt die Existenz einer Produktion $B \rightarrow \psi_{k-1} A C \tau$ in P , mit $C \xrightarrow{+} I \lambda$, $\tau, \lambda \in \Gamma^*$
4. Aus $\text{End}(\psi_{k-1}) \triangleleft A$ und $A \dot{=} I$ folgt die Existenz einer Produktion $B \rightarrow A \tau$ in LK , $\tau \in \Gamma^*$
5. Aus $\text{End}(\psi_{k-1}) \triangleleft A$ und $A \triangleleft I$ folgt die Existenz einer Produktion $B \rightarrow A C \tau$ in LK mit $C \xrightarrow{+} I \tau$, $\tau, \lambda \in \Gamma^*$

Im Fall von $\text{End}(\psi_{k-1}) \triangleleft A$ und $A \triangleright I$ soll A nicht zur Menge LRK gehören. Damit wird erreicht, daß keine Variablen eingesetzt werden, welche sofort reduziert werden müssen.

Ist die Menge LRK bestimmt, so sind drei Fälle möglich

1. $|LRK| = 1$
2. $|LRK| = 0$
3. $|LRK| > 1$

Im ersten Fall wird die eindeutig bestimmte Variable A mit $LRK = \{A\}$ an Stelle des Kandidaten gekellert, und mit diesem neuen Kellerinhalt die Analyse fortgesetzt.

Im zweiten und dritten Fall wird ein neuer Kandidat durch Erweiterung des alten Kandidaten nach links oder nach rechts bestimmt. Auf eine Beschreibung des Algorithmus, mit Hilfe dessen die Richtung der Erweiterung bestimmt wird, können wir aus zwei Gründen verzichten:

Leinius gibt keine Erklärungen dazu, wie die Kriterien mit Hilfe derer er die Erweiterungsrichtung bestimmt, abhängen von der Fehlersituation;

Der Algorithmus beeinflusst die Qualität der Fehlerbehandlung nur unwesentlich, da in den meisten, in der Praxis auftretenden Fällen eine Anwendung des Algorithmus nicht nötig ist.

Beispiel 3.4: Die Methode von Leinius

Bei der Analyse der Zeichenfolge aus Beispiel 3.3 wird beim Lesen von *begin* der erste Fehler erkannt. Der Kellerinhalt dabei ist

$\ast Ph \text{ var } Il1 \text{ id}$

Wegen der Gültigkeit der Relationen: $\ast \triangleleft Ph$, $Ph \triangleleft var$, $var \triangleleft Il1$, $Il1 \triangleleft id$ ist die Zeichenfolge *var Il1 id* der erste Kandidat. Die Menge LRK besteht aus der Variablen *V*. Deshalb wird die Analyse mit dem Kellerinhalt $\ast Ph V$ fortgesetzt. Der nächste Fehler wird beim Kellern von *I* entdeckt. Der Kellerinhalt ist dabei

$\ast Ph V \text{ begin id} : G \text{ id } [(E$

Das aktuelle Zeichen ist *+*.

Die Zeichenfolge $(E]$ ist der erste Kandidat.

Die Menge LRK besteht aus der Variablen *T* und die Analyse wird mit dem Kellerinhalt

$Ph V \text{ begin id} : G \text{ id } [T$

fortgesetzt. Deswegen wird beim Lesen von *end* ein weiterer Fehler entdeckt, wobei der Kellerinhalt

$\ast Ph V \text{ begin id} : G \text{ id } [E$

ist.

Der erste Kandidat ist die Zeichenfolge $id \in E$. Die Menge LRK besteht aus der Variablen E , und die Analyse wird mit dem Kellerinhalt

*Ph V begin $id : G E$

mit dem aktuellen Zeichen end fortgesetzt. Dabei tritt kein weiterer Fehler auf.

Das heißt, der Algorithmus von Leinius meldet die selben Fehler wie der Algorithmus von Wirth. Insbesondere wird ein Folgefehler gemeldet.

Δ

Insgesamt können wir das Verfahren von Leinius wie folgt beurteilen:

Die Vorteile sind:

1. Die Verwendung von Links- und Rechtskontext der Fehlerstelle zur Auswahl der Korrektur liefert bei richtig gewählten Kandidaten sehr gute Ergebnisse.
2. Das Verfahren ist vollständig unabhängig von der speziellen Sprache, d.h. automatisch erzeugbar.
3. Das Verfahren behandelt jede Eingabezeichenfolge vollständig, da durch wiederholte Erweiterung des Kandidaten stets ein Kandidat gefunden wird, der eine eindeutige Korrektur erlaubt.
4. Der Aufwand der Implementierung ist vertretbar, wenn auch wesentlich größer als bei dem Verfahren von Wirth.
5. Die Analyse korrekter Zeichenfolgen wird durch die Existenz der Fehlerbehandlung nicht beeinflusst.

Die Nachteile sind:

1. Die Kandidatenwahl, insbesondere die des Anfangskandidaten ist zu wenig durch die spezielle Fehlersituation gesteuert. Deswegen treten häufig Folgefehler auf.
2. Die Beschränkung der Korrekturmöglichkeiten auf Ersetzen des Kandidaten durch eine Variable wird zahlreichen Fehlersituationen nicht gerecht, und ist deshalb eine weitere Ursache für Folgefehler. Zum Beispiel ist in vielen Fällen das Streichen des Kandidaten eine adäquate Korrektur.

3.2.3 Die Methode von Rhodes

Die Methode von Rhodes ist in den von der Analysefunktion F_3 (siehe Satz 2.2) induzierten Analysator A_{F_3} eingebettet. Wie bereits in Abschnitt 2.2.3 erläutert, erkennt A_{F_3} die mit Zeichenpaar-, Reduktions-, Kellerungs- und RS-Fehler bezeichneten Fehlertypen. Die Methode von Rhodes unterscheidet sich von der von Leinius in Kandidatenwahl und Korrektur. Die Kandidatenwahl wird mit Hilfe der sogenannten Kondensationsphase durchgeführt, deren Sinn es ist, den Kontext der Fehlerstelle zu verdichten, um dadurch eine bessere Kandidatenwahl zu erreichen.

Die Kondensationsphase wird in drei Schritten durchgeführt: dem ersten Rückwärtsschritt, dem Vorwärtsschritt und dem zweiten Rückwärtsschritt.

Beim ersten Rückwärtsschritt wird zunächst ein spezielles Zeichen, im folgenden mit $?_1$ bezeichnet, als oberstes Element in den Keller eingetragen, welches den Fehlermarkierungspunkt eins repräsentiert. Das heißt, der Kellerinhalt kann jetzt in der Form $\psi_0 \dots \psi_k ?_1$, wo für $\psi_0 \dots \psi_k$ die Forderungen 1, 2 und 4 aus Satz 2.3 gelten, dargestellt werden. Das aktuelle Zeichen sei I . Im Fall eines Zeichenpaarfehlers ist I ein terminales Zeichen und es gilt $\text{End}(\psi_k) \odot I$. Im Fall eines Kellerungsfehlers ist I eine Variable und es gilt $\text{End}(\psi_k) \odot I$. Im Fall eines Reduktions- oder eines RS-Fehlers ist I ein terminales Zeichen. Der erste Rückwärtsschritt besteht darin, daß zwischen $\text{End}(\psi_k)$ und $?_1$ die Relation $>$ als gültig angenommen wird und alle unter dieser Annahme möglichen Reduktionen des Kellerinhalts links von $?_1$ durchgeführt werden. Selbstverständlich ist eine Reduktion nur dann möglich, wenn sie keinen Kellerungs- bzw. RS-Fehler verursacht.

Für den Vorwärtsschritt wird zwischen $?_1$ und I die Relation $\triangleleft$ als gültig angenommen und I gekellert. Jetzt wird mit Hilfe des Analysealgorithmus (ohne Prüfung auf RS-Fehler) die Eingabezeichenfolge solange weiteranalysiert, bis erneut Fehler entdeckt werden (diese Situation muß, wegen $?_1 \triangleleft I$ für alle Zeichen I des zugrunde liegenden Alphabets einmal eintreten).

Beim zweiten Rückwärtsschritt wird zunächst ein weiteres spezielles Zeichen, im folgenden mit $?_2$ bezeichnet, als oberstes Element in den Keller

eingetragen, welches den Fehlermarkierungspunkt zwei repräsentiert. Das heißt, jetzt kann der Kellerinhalt in der Form $\psi_0 \psi_1 \dots \psi_k ?_1 \psi_{k+1} ?_2$ dargestellt werden. Der zweite Rückwärtsschritt besteht darin, daß alle, unter der Annahme der Gültigkeit von $\text{End}(\psi_{k+1}) > ?_1$ möglichen Reduktionen von ψ_{k+1} vorgenommen werden.

Sei nach dem zweiten Rückwärtsschritt der Kellerinhalt durch $\psi_0 \psi_1 \dots \psi_k ?_1 \psi_{k+1} ?_2$ gegeben, dann definiert Rhodes mit:

$$\begin{aligned} \text{Kand 1} &:= \psi_{k+1}, \\ \text{Kand 2} &:= \psi_k, \\ \text{Kand 3} &:= \psi_k \psi_{k+1}, \end{aligned}$$

drei Kandidaten für eine mögliche Korrektur.

Beispiel 3.5: Kondensationsphase bei Rhodes

Für die Zeichenfolge aus Beispiel 3.3 ergibt sich bei Fehlererkennung der Kellerinhalt

$$\ast Ph \text{ var } IL1 \text{ id } ?_1$$

Im ersten Rückwärtsschritt wird dieser reduziert zu

$$\ast Ph \text{ Vh } ?_1$$

Der Vorwärtsschritt endet mit der Erkennung des zweiten Fehlers. Der Kellerinhalt dabei ist

$$\ast Ph \text{ Vh } ?_1 \text{ begin id : } G \text{ id } [(E) ?_1$$

Der zweite Rückwärtsschritt ergibt keine Veränderungen des Kellers, also erhalten wir die folgenden Kandidaten:

$$\begin{aligned} \text{Kand 1} &= \text{begin id : } G \text{ id } [(E) \\ \text{Kand 2} &= \text{Vh} \\ \text{Kand 3} &= \text{Vh begin id : } G \text{ id } [(E) \end{aligned}$$

Δ

In der Korrekturphase bestimmt Rhodes zunächst für jeden der drei Kandidaten die Menge der lokal syntaktisch korrekten Korrekturen. Als mögliche Korrekturen eines Kandidaten sind seine Löschung oder seine Erset-

zung durch eine rechte Seite einer Produktion zugelassen. Eine Korrektur heißt lokal syntaktisch korrekt, wenn ihre Ausführung nach Wiederaufnahme der Analyse nicht unmittelbar neue Fehler verursacht.

Aus der Menge der möglichen Korrekturen wird die durchzuführende, mit Hilfe eines gewichteten Mustervergleichs-Verfahrens bestimmt. Es wird für jede lokal syntaktisch korrekte Korrektur eine Zahl berechnet, welche die Kosten der Korrektur repräsentiert. Eine Korrektur mit den niedrigsten Kosten wird dann durchgeführt. Im Fall einer Korrektur des zweiten Kandidaten werden nach der Korrektur die Fehlermarkierungspunkte $?_1$ und $?_2$ gelöscht und die Analyse mit dem ersten Zeichen des Kandidaten eins als aktuelles Zeichen fortgesetzt, d.h. der Kandidat eins wird jetzt als Teil der Eingabezeichenfolge aufgefaßt. Im Fall einer Korrektur des ersten oder dritten Kandidaten werden ebenfalls, soweit noch vorhanden, die Fehlermarkierungspunkte gelöscht und die Analyse mit dem nächsten Zeichen des Eingabemediums als aktuelles Zeichen fortgesetzt. Sind die Kosten für jede mögliche Korrektur größer als eine bestimmte vorgegebene obere Schranke, so ist das Verfahren gescheitert. Das heißt, alle Korrekturen des Kellers sind so tiefgreifend, daß ihre Durchführung als nicht sinnvoll erscheint, weil sie wahrscheinlich zu viele Folgefehler nach sich ziehen würde.

Die Kosten für eine Korrektur werden wie folgt berechnet.

Für jedes Zeichen X des Alphabets der zugrunde liegenden Sprache wird festgelegt, welche Kosten eine der elementaren Korrekturen, Einschlebung (I(X)), Weglassung (D(X)) bzw. Ersetzung von X durch jedes Zeichen Y des Alphabets (R(X,Y)) verursacht. Jede Korrektur kann man sich aus einer minimalen Folge von elementaren Korrekturen zu Stande gekommen denken. Die Kosten für die gesamte Korrektur ergibt sich dann aus der Summation der Kosten der elementaren Korrekturen der minimalen Folge. Existieren mehrere minimale Folgen, so definiert die Folge mit den kleinsten Gesamtkosten die Kosten der Korrektur.

Zur Wahl der Kosten der elementaren Korrekturen macht Rhodes keine systematischen Aussagen. Er beschränkt sich auf die Angabe einiger Prinzipien, die seiner Erfahrung nach gute Ergebnisse liefern.

Solche sind:

1. Stets soll $R(X,Y) < D(X) + I(Y)$ gelten.
2. Für die Klammern $(,), [,]$ sollen die Kosten für Einschubung höher sein als für die Auslassung.
3. Bei Sprachen mit Blockstruktur, wo diese durch Schlüsselwörter gekennzeichnet ist, soll die Weglassung von Schlüsselwörtern, welche die Blockstruktur beenden, verboten sein (Kosten höher als das Maximum).
4. Korrekturen, welche weitreichende Konsequenzen auf die Struktur eines Wortes haben, sollen höhere Kosten verursachen, als solche, die das nicht tun. Zum Beispiel soll $I([) > I(\text{identifizier})$ gelten.

Beispiel 3.6: Korrektur nach Rhodes

Von den drei Kandidaten aus Beispiel 3.5 existiert nur für Kand 2 eine lokal syntaktisch korrekte Korrektur (Ersetzung von Vh durch V). Nach Fortsetzung der Analyse wird beim Kellerinhalt $\#Ph V \text{ begin id: } G \text{ id } [(E]$ ein Reduktionsfehler erkannt. Die Kondensationsphase liefert die Kandidaten:

Kand 1 = F
 Kand 2 = $(E]$
 Kand 3 = $(E] F$

Für Kand 1 existiert keine lokal syntaktisch korrekte Korrektur. Für Kand 2 und Kand 3 ist eine Ersetzung durch F oder T lokal syntaktisch korrekt. Die niedrigsten Kosten, nämlich:

$$D([) + I())$$

hat die Ersetzung von Kand 2 durch F .

Nach Durchführung dieser Korrektur wird die Analyse mit dem Kellerinhalt

$\#Ph V \text{ begin id : } G \text{ id } [F$

und dem aktuellen Zeiten + fortgesetzt. Wie bei den früheren Methoden wird beim Verarbeiten von *end* und Kellerinhalt

$\#Ph V \text{ begin id : } G \text{ id } [F$

ein weiterer Fehler erkannt. Dieser wird durch Ersetzen von $id [E$ durch F behoben.

Das heißt, auch die Fehlerbehandlung von Rhodes meldet einen Folgefehler bei der Analyse der Zeichenfolge aus Beispiel 3.3.

Δ

Zusammenfassend ergibt sich folgende Wertung des Verfahrens:

Die Vorteile sind:

1. Die Kondensationsphase liefert im allgemeinen Kandidaten, welche eine sinnvolle Korrektur zulassen.
2. Das Verfahren ist bis auf die Wahl der Kosten unabhängig von der speziellen Grammatik, also im wesentlichen automatisch erzeugbar.
3. Mit Hilfe der Gewichtungen der Kosten können benutzergruppenspezifisch auftretende Fehler besonders gut behandelt werden.
4. Der Aufwand für die Implementierung des Verfahrens ist vertretbar.
5. Die Analyse korrekter Zeichenfolgen wird durch die Existenz der Fehlerbehandlung nicht beeinflusst.

Die Nachteile sind:

1. Es gibt bisher keine systematischen Methoden zur Wahl der Kostenvektoren in Abhängigkeit der speziellen Grammatik.
2. Das Verfahren behandelt nicht jede Eingabezeichenfolge vollständig. Bei gewissen 'groben Fehlern' wird die Analyse abgebrochen.
3. Das Problem der Vermeidung von Folgefehlern ist nur unbefriedigend gelöst. Die Gründe dafür liegen sowohl bei der immer noch zu lokalen Kandidatenwahl, als auch bei der Wahl der Korrektur mit Hilfe der willkürlichen Gewichte für die Kosten.

3.2.4 Zusammenfassende Wertung

Zusammenfassend erhalten wir folgende Wertung der bisher bekannten Methoden:

Jede der Methoden liefert nach spezifischer Anpassung an die spezielle Sprache Ergebnisse, welche für Anwendungen in der Praxis zwar brauchbar,

aber nicht optimal sind.

Keine der Methoden genügt allen in Abschnitt 0 an eine Fehlerbehandlung gestellten Forderungen. Insbesondere erzeugt jede der Methoden Folgefehler. Über die Art der Folgefehler existieren keine systematischen Aussagen.

Der Grund für die Folgefehler liegt darin, daß alle Methoden weder für die Kandidatenwahl noch für die Wahl der Korrektur genügend Kontext der Fehlerstelle betrachten um entscheiden zu können, welcher Fehler vorliegt. Um dies zu verdeutlichen, betrachten wir ein weiteres Beispiel.

Beispiel 3.7: Folgefehler bei den Methoden von Wirth, Leinius und Rhodes

Sei die fehlerhafte L1-Zeichenfolge

```
program id ;  
begin  
    id := id [ ( id + id ] + id ) * id ]  
end.  
                                †
```

gegeben. Sie enthält einen Fehler, die erste schließende Klammer (durch † markiert) ist überflüssig. Jeder der drei Analysealgorithmen meldet bei der Analyse der obigen Zeichenfolge drei Fehler, also zwei Folgefehler. Der Grund dafür ist, daß die Fehler stets unabhängig vom rechten Kontext korrigiert werden.

Δ

An obigem Beispiel erkennt man, daß zur Diagnose von Fehlern im allgemeinen beliebig großer Links- bzw. Rechtskontext herangezogen werden muß. (Erst nach dem Lesen der letzten schließenden Klammer kann der Fehler korrekt diagnostiziert werden.) Deswegen verursachen alle Fehlerbehandlungsmethoden, welche auf einer Korrektur der Eingabezeichenfolge bzw. des Kellerinhalts beruhen und nicht die Möglichkeit haben, beliebig großen Kontext der Fehlerstelle zur Fehlerdiagnose heranzuziehen, Folgefehler. Aus diesem Grund wird im folgenden Abschnitt eine Fehlerbehandlungsmethode entwickelt, die nicht mehr auf einer Korrektur der Eingabezeichenfolge

bzw. des Kellers basiert. Ein Vorteil dieser Methode wird sein, daß sie keine Folgefehler erzeugt, ein Nachteil wird sein, daß sie nicht alle Fehler erkennt.

3.3 Nachsilbenanalyse als Fehlerbehandlung für einfache Präzedenzsprachen

Die Nachsilbenanalyse als Fehlerbehandlungsmethode, welche keine Folgefehler produziert, wurde von P. Gonser⁽¹⁾ vorgeschlagen. In diesem Abschnitt wird ein Algorithmus entworfen, welcher die Nachsilbenanalyse für einfache Präzedenzsprachen realisiert.

3.3.1 Beschreibung und Diskussion des Prinzips der Nachsilbenanalyse

Unter "Nachsilbenanalyse" verstehen wir die folgende Fehlerbehandlungsmethode: Wird in der Eingabezeichenfolge ein Fehler entdeckt, so wird nicht mehr versucht, diesen Fehler zu korrigieren um dadurch die Analyse der Eingabezeichenfolge fortsetzen zu können, sondern es wird festgestellt, ob der noch zu analysierende Teil der Eingabezeichenfolge ein Suffix der Sprache ist. Diese Strategie wird solange angewendet, bis die Eingabezeichenfolge vollständig analysiert ist. Die Folge der dabei gefundenen Fehlerstellen bezeichnen wir als Nachsilbenanalyse.

Definition 3.8

Sei L formale Sprache über dem Alphabet Σ , $x \in \Sigma^* \setminus L$ mit $x = a_1 \dots a_n$, $n \in \mathbb{N}$, dann gilt:

1. $s \in \mathbb{N}^p$ mit $s = (j_1, \dots, j_p)$ heißt eine Nachsilbenanalyse von x (bzgl. L) der Länge p , wenn gilt:

(1) P. Gonser hat die Nachsilbenanalyse im Rahmen des Oberseminars von Prof. Seegmüller im SS 1975 vorgeschlagen.

1.1 $1 \leq j_i < j_{i+1} < n+1$ für $i = 1, 2, \dots, p-1$

1.2 Im Fall von $j_1 > 1$ gilt: $a_1 \dots a_{j_1-1} \in \text{Präfix}(L)$

1.3 Für alle $k \in \{1, \dots, p-2\}$ mit $j_{k+1} < j_{k+1}-1$ gilt:

$$a_{j_k+1} \dots a_{j_{k+1}-1} \in \text{Infix}(L).$$

1.4 Im Fall von $j_p < n$ gilt: $a_{j_{p-1}+1} \dots a_{j_p-1} \in \text{Infix}(L) \wedge$

$$a_{j_p+1} \dots a_n \in \text{Suffix}(L)$$

Im Fall von $j_p = n$ gilt: $a_{j_{p-1}+1} \dots a_{n-1} \in \text{Infix}(L)$

Im Fall von $j_p = n+1$ gilt: $a_{j_{p-1}+1} \dots a_n \in \text{Infix}(L) \setminus \text{Suffix}(L)$

2. Eine Nachsilbenanalyse s von x heißt minimal, wenn es keine Nachsilbenanalyse t von x gibt mit kleinerer Länge.

Beispiel 3.8: Eine Nachsilbenanalyse

Die bezüglich L_1 fehlerhafte Zeichenfolge x sei durch

```

x :=  program id ;
      begin
          id := id [ ( id + id ) + id * id ;
          id ;                               ↑F1           ↑F2 (1)

          while id then id := id ) ;
          ↑F3           ↑F4

          if id := id + id then id = id ;
          ↑F5           ↑F6

          end

      end.
      ↑F7
    
```

gegeben. Sie enthält sieben Fehler. Die "fehlerhaften" Zeichen sind mit

(1) Im Sinne unseres Fehlerbegriffs, muß F2 hier und nicht erst beim nächsten ; markiert werden.

†Fi markiert. $s = (14, 21, 24, 29, 33, 40)$ ist eine minimale Nachsilbenanalyse von x der Länge 6. Die Infixe von L sind:

```

program id ; begin id := id [ ( id + id
+ id * id ; id
while id
id := id
; if id
= id + id then id
id ; end end.

```

Δ

Es wäre wünschenswert, wenn für jede minimale Nachsilbenanalyse einer Zeichenfolge x die Länge der Nachsilbenanalyse identisch wäre mit der Anzahl der Fehler in x (bzgl. des in Abschnitt 3.1 definierten Fehlerbegriffs).

Daß dies im allgemeinen nicht der Fall ist, sieht man am Beispiel 3.8. Es wird nicht erkannt, daß ein Zeichen *end* überflüssig ist (F7). Allgemein gilt, daß fehlende oder überflüssige schließende "Klammern" (1) nicht erkannt werden, wenn zwischen den öffnenden und den schließenden Klammern ein Fehler entdeckt wird. D.h. es gibt Fälle, in denen mit Hilfe einer Nachsilbenanalyse nicht alle Fehler erkannt werden. Im folgenden Satz wird gezeigt, daß die Anzahl der mit Hilfe einer minimalen Nachsilbenanalyse diagnostizierten Fehler stets kleiner oder gleich der Anzahl der Fehler in x ist.

Satz 3.1

Sei L formale Sprache über dem Alphabet Σ , $x \in \Sigma^* \setminus L$, x enthalte $k \geq 1$ Fehler, s sei eine minimale Nachsilbenanalyse von x , dann gilt:

$$|s| \leq k$$

(1)

Der Begriff Klammer wird als bekannt vorausgesetzt.

Beweis:

Um die Behauptung zu beweisen ist es hinreichend zu zeigen, daß eine Nachsilbenanalyse existiert, für welche die Behauptung gilt.

$x = a_1 \dots a_n$ enthält k Fehler. Deshalb existiert eine Folge von Fehlern f_i , $i = 1, \dots, k$, welche eine Fehlerdiagnose darstellt. Jeder Fehler f_i definiert eine Fehlerstelle j_i .

O.B.d.A. kann vorausgesetzt werden, daß $1 \leq j_i \leq j_{i+1} \leq n+1$ gilt.

Sei $t_q \in \mathbb{N}$, $q = 1, \dots, m$, $m \leq k$ eine Folge mit den Eigenschaften:

$$1. \{t_q \mid q=1, \dots, m\} = \{j_i \mid i = 1, \dots, k\}$$

$$2. t_q < t_{q+1} \quad q = 1, \dots, m-1$$

Dann ist im Fall vom $t_{m-1} = n$ und $t_m = n+1$ die Folge t_q , wo $q = 1, \dots, m-1$ und sonst die Folge t_q , wo $q=1, \dots, m$ eine Nachsilbenanalyse von x .

Der Beweis dieser Behauptung ergibt sich aus der Definition der Folge f_i , denn durch die mit Hilfe der Fehler f_i an den Korrekturstellen j_i definierten Korrekturen wird die Eingabezeichenfolge x auf ein Wort y der Sprache abgebildet. Daraus folgt, daß die Zeichenfolgen

$a_1 \dots a_{j_1-1}$, $a_{j_1+1} \dots a_{j_2-1}$, usw., soweit sie existieren, Präfixe, Infixe bzw. Suffixe von L sind. Daraus folgt die Behauptung.

q.e.d.

An dem obigen Beweis erkennt man eine weitere Schwäche der Nachsilbenanalyse. In Fällen, wo sich an einer Stelle der Eingabezeichenfolge mehr als ein Fehler befindet, wird nur ein Fehler gemeldet.

Beispiel 3.9: Grobe Fehler

Bei der Analyse der bezüglich L_1 fehlerhaften Zeichenfolge

```
program id;  
begin  
    id := id; id := id;  
program id;  
  ↑  
begin  
    id := id ; id := id;  
end.
```

wird genau ein Fehler gemeldet (beim Lesen des mit † markierten Zeichens). Die Zeichenfolge enthält jedoch vier Fehler.

Δ

Insgesamt ergibt sich, daß die Erstellung von minimalen Nachsilbenanalysen für fehlerhafte Eingabezeichenfolgen keine Fehlerbehandlungsmethode ist, welche die Forderung E1 aus Abschnitt 0.1 vollständig erfüllt.

Ihre Nachteile sind:

1. Fehler in "klammerartigen" Strukturen werden nicht erkannt, wenn zwischen öffnender und schließender Klammer ein zusätzlicher Fehler ist (Bsp. 3.8).
2. Bei groben Fehlern diagnostizieren die Fehlermeldungen die Art des Fehlers i.allg. nicht ausreichend (Bsp. 3.9).

Ihre Vorteile sind:

1. Durch minimale Nachsilbenanalysen werden keine Folgefehler ⁽¹⁾ gemeldet, die Probleme der Kellerkorrektur treten nicht auf (Satz 3.1).
2. Jede Eingabezeichenfolge wird zu Ende analysiert, ohne daß Zeichen überlesen werden.

Wie stark sich die prinzipiellen Schwächen der Nachsilbenanalyse in der Praxis auswirken, kann nur empirisch festgestellt werden, da dies davon abhängt, wie häufig Fehlersituationen auftreten, welche mit Hilfe der Nachsilbenanalyse nur unzureichend diagnostiziert werden können. Erfahrungsgemäß gilt, daß mindestens 80 Prozent der auftretenden Fehler Einsymbolfehler sind (siehe [Dam.64]), so daß z.B. die unter zweitens aufgeführte Schwäche für praktische Anwendungen nur geringe Bedeutung hat.

Im nächsten Abschnitt werden theoretische Grundlagen erarbeitet, mit deren Hilfe in den weiteren Abschnitten fehlerbehandelnde Analysatoren, welche das Prinzip der Nachsilbenanalyse realisieren, gewonnen werden.

⁽¹⁾ Wie früher definiert, sind syntaktische Folgefehler gemeint.

3.3.2 Theoretische Grundlagen für die Realisierung fehlerbehandelnder Analysealgorithmen, welche Nachsilbenanalysen produzieren

In diesem Abschnitt wird das Problem der Konstruktion eines fehlerbehandelnden Analysators, welcher jede fehlerhafte Eingabezeichenfolge auf eine minimale Nachsilbenanalyse abbildet, zurückgeführt auf das Problem der Konstruktion von Analysatoren mit LP-Eigenschaft.

Definition 3.9

Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik, $L = L(G)$, $a \in \Sigma$, dann gilt:

1. Unter der Nachsilbe $NS_L.a$ von a (bezüglich L) verstehen wir die durch

$$NS_L.a := \{y \in \Sigma^* \mid \exists z \in \Sigma^* : zy \in L \wedge a \in \text{Anf}(y)\}$$

definierte Sprache über Σ .

2. Unter der durch L definierten Nachsilbensprache $NS.L$ verstehen wir die durch

$$NS.L := \{x \in \Sigma^* \mid \exists z \in \Sigma^* : zx \in L\}$$

definierte Sprache über Σ .

3. Unter einem Nachsilbenanalysator $NSA_L^{(1)}$ für L verstehen wir jeden Analysator NSA_L für $NS.L$ mit $FM = \mathbb{N}$, für den für $x = a_1 \dots a_n$ gilt:

$$3.1 \quad NSA_L(x) = k \leq n \quad \} \quad a_1 \dots a_k \notin \text{Präfix}(NS_L.a_1)$$

$$3.2 \quad NSA_L(x) = n + 1 \quad \} \quad a_1 \dots a_n \in \text{Präfix}(NS_L.a_1) \setminus NS_L.a_1$$

4. Ein Nachsilbenanalysator NSA_L (für L) hat genau dann die LP-Eigenschaft (für L) wenn gilt:

$$NSA_L(a_1 \dots a_n) = k \quad \} \quad a_1 \dots a_{k-1} \in \text{Präfix}(NS_L.a_1)$$

Definition 3.10

Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik, $L = L(G)$, A ein Analysa-

⁽¹⁾ Für NSA_L bzw. $NS_L.a$ schreiben wir oft NSA bzw. $NS.a$ (siehe Fußnote S.14).

tor und NSA ein Nachsilbenanalysator für L , $x = a_1 \dots a_n \in \Sigma^+ \setminus L$, dann gilt:

Die Folge $q_i \in \mathbb{N}$, $i = 1, \dots, p$, $p \in \mathbb{N}$, definiert durch

$$q_1 := A(x)$$

und

$$q_{i+1} := q_i + \text{NSA}(a_{q_i+1} \dots a_n) \text{ falls } q_i < n$$

heißt die durch A und NSA definierte Zerlegung von x .

Satz 3.2

Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Sprache, $L = L(G)$, A ein Analysator und NSA ein Nachsilbenanalysator für L , $x = a_1 \dots a_n \in \Sigma^+$, dann gilt:

Die Abbildung $\mathcal{U}_{A, \text{NSA}}$ definiert durch

$$\mathcal{U}_{A, \text{NSA}}(x) := \begin{cases} \text{true} & \text{falls } x \in L \\ (q_1, \dots, q_p) & \text{falls } x \notin L \text{ und} \\ & q_i, i=1, \dots, p \text{ die durch } A \text{ und } \text{NSA} \\ & \text{definierte Zerlegung von } x \text{ ist} \end{cases}$$

ist ein fehlerbehandelnder Analysator für L , wobei $\mathbb{N}$ die Menge der Fehlermeldungen ist.

Der Beweis von Satz 3.2 ist eine triviale Folgerung der Definitionen 3.7, 3.9 und 3.10.

Definition 3.11

Der in Satz 3.2 konstruierte Analysator $\mathcal{U}_{A, \text{NSA}}$ heißt der von A und von NSA induzierte (fehlerbehandelnde) Analysator für L .

Die wichtigste Eigenschaft der oben definierten induzierten Analysatoren ist, daß für den Fall, daß die zugrunde liegenden Analysatoren die LP-Eigenschaft haben, der induzierte Analysator fehlerhafte Eingabezeichenfolgen auf minimale Nachsilbenanalysen abbildet.

Satz 3.3

Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik, $L = L(G)$, A ein Analysa-

tor und NSA ein Nachsilbenanalysator für L, jeweils mit LP-Eigenschaft.

$\mathcal{U}_{A,NSA}$ sei der von A und NSA induzierte Analysator, dann gilt für $x \in \Sigma^+ \setminus L$:

$\mathcal{U}_{A,NSA}^{(x)} = (k_1, \dots, k_p)$ ist minimale Nachsilbenanalyse von x bzgl. L.

Beweis:

Haben A und NSA die LP-Eigenschaft, so ist es trivial, daß die Zerlegung $(k_1, \dots, k_p)$ eine Nachsilbenanalyse ist. Zu zeigen ist nur deren Minimalität.

Sei $(k_1, \dots, k_p)$ keine minimale Nachsilbenanalyse von x.

Dann existiert eine Nachsilbenanalyse $(j_1, \dots, j_q)$ von x mit $q < p$.

Für $i = 1, \dots, q$ beweisen wir durch vollständige Induktion über i daß gilt: $j_i \leq k_i$

Induktionsanfang: $i = 1$

Da A die LP-Eigenschaft hat, ist $a_1 \dots a_{k_1-1}$ das längste Präfix von x bzgl. L. Da $a_1 \dots a_{j_1-1}$ ebenfalls ein Präfix von L sein muß, folgt $j_1 \leq k_1$.

Induktionsschritt:

Die Behauptung ist für $i = m + 1$ zu zeigen, unter der Voraussetzung, daß sie für $i \leq m \leq q$ gilt.

$$\begin{aligned}
 j_{m+1} > k_{m+1} \quad \Bigg\} \quad & u := a_{j_{m+1}} \dots a_{k_m} \dots a_{j_{m+1}-1} \in \text{Präfix}(\text{NS}.a_{j_{m+1}}) \quad \Bigg\} \\
 & \exists y \in \Sigma^* : uy \in \text{NS}.a_{j_{m+1}} \quad \Bigg\} \\
 & uy \in \text{Suffix}(L) \quad \Bigg\} \\
 & \bar{a}_{k_m+1} \dots a_{k_{m+1}} \dots a_{j_{m+1}-1} y \in \text{Suffix}(L) \quad \Bigg\} \\
 & a_{k_m+1} \dots a_{k_{m+1}} \dots a_{j_{m+1}-1} \in \text{Präfix}(\text{NS}.a_{k_{m+1}})
 \end{aligned}$$

Das ist jedoch ein Widerspruch zur Konstruktion von $(k_1, \dots, k_p)$. Das heißt, aus der Induktionsvoraussetzung folgt $j_{m+1} \leq k_{m+1}$. Daraus folgt,

daß $a_{q+1} \dots a_{k+1} \dots a_n \in \text{NS}.a_{j+1}$ gilt. Dies ist ebenfalls ein Widerspruch zur Konstruktion von $(k_1, \dots, k_p)$, d.h., die Annahme, daß $(k_1, \dots, k_p)$ keine minimale Nachsilbenanalyse ist, hat zum Widerspruch geführt.

q.e.d.

Durch den Satz 3.3 ist die Konstruktion von minimalen Nachsilbenanalysen zurückgeführt auf die Konstruktion von Analysatoren und Nachsilbenanalysatoren, jeweils mit LP-Eigenschaft. Das heißt, um das Problem der Konstruktion von minimalen Nachsilbenanalysatoren für einfache Präzedenzsprachen zu lösen, wird im folgenden ein Nachsilbenanalysator mit LP-Eigenschaft für den Fall konstruiert, daß die Ausgangssprache eine einfache Präzedenzsprache ist. Um dieses Problem abzugrenzen, untersuchen wir zunächst allgemein, welche Zusammenhänge zwischen der Sprachklasse der Ausgangssprache und den Sprachklassen der Nachsilben bestehen.

Satz 3.4

Sei $G = (N, \Sigma, P, S)$ eine kontextfreie Grammatik, $L = L(G)$, dann gilt:
Für $a \in \Sigma$ ist $\text{NS}_L.a$ eine kontextfreie Sprache.

Der Satz stammt von P. Gonser, er wird durch Konstruktion einer Grammatik für $\text{NS}.a$ bewiesen. Eine Möglichkeit einer solchen Konstruktion kann bei Springer [Spr.76] nachgelesen werden.

Durch den obigen Satz ist bewiesen, daß Analysatoren für Nachsilbensprachen von kontextfreien Sprachen existieren. Eine günstige Voraussetzung für eine effiziente Implementierung der Nachsilbenanalyse wäre es, wenn die Nachsilbensprachen von derselben Sprachklasse wären wie die zugrunde liegende Sprache. Daß dies im allgemeinen nicht der Fall ist zeigt der folgende Satz.

Satz 3.5

Zu jeder der drei folgenden Eigenschaften existiert ein Alphabet Σ und eine formale Sprache L über Σ , welche diese Eigenschaft besitzt:

1. L ist LL(1)⁽¹⁾ und es existiert ein $a \in \Sigma$, für das NS.a nicht deterministisch, also nicht LL(1) ist.
2. L ist LR(1) und es existiert ein $a \in \Sigma$, für das NS.a nicht deterministisch, also nicht LR(1) ist.
3. L ist eine einfache Präzedenzsprache und es existiert ein $a \in \Sigma$, so daß NS.a keine eindeutige, also auch keine einfache Präzedenzsprache ist.

Beweis:

Zu (1) und (2):

Die Sprache $L := \{ax^n1^n \mid n \geq 1\} \cup \{bx^n1^{2n} \mid n \geq 1\}$ ist LL(1), das heißt auch LR(1).

$NS.x = \{x^n1^n \mid n \geq 1\} \cup \{x^n1^{2n} \mid n \geq 1\}$ ist nicht deterministisch (siehe [Aho.U11,72], Seite 717).

q.e.d.

Zu (3):

Die Sprache $L := \{yx a^n b^n c^k \mid k, n \in \mathbb{N}\} \cup \{zx a^k b^n c^n \mid k, n \in \mathbb{N}\}$ ist eine einfache Präzedenzsprache.

Produktionen, welche eine einfache Präzedenzgrammatik für L definieren, sind:

$$\begin{aligned}
 S &\rightarrow yx A X \mid B Y \\
 A &\rightarrow A1 A b \mid A1 b \\
 X &\rightarrow C1 X \mid C1 \\
 B &\rightarrow B a \mid z x C \\
 Y &\rightarrow B1 Y c \mid A1 c \\
 C &\rightarrow A1 \\
 A1 &\rightarrow a \\
 B1 &\rightarrow b \\
 C1 &\rightarrow c
 \end{aligned}$$

⁽¹⁾ Die Begriffe LL(1), LR(1), deterministisch, eindeutig seien definiert wie bei [Aho.U11,72]

$NS.x = \{xa^n b^n c^k \mid k, n \in \mathbb{N}\} \cup \{xa^k b^n c^n \mid k, n \in \mathbb{N}\}$ ist eine mehrdeutige Sprache ([Mau,69], Seite 188).

Aus den beiden vorangegangenen Sätzen folgt, daß für Nachsilbensprachen von kontextfreien Sprachen Analysatoren existieren und daß diese im allgemeinen nicht deterministisch sind. Im nächsten Abschnitt wird gezeigt, daß in dem Fall, in welchem die zugrunde liegende Sprache eine einfache Präzedenzsprache ist, für die Nachsilbenanalysatoren dennoch effiziente Implementierungen möglich sind.

3.3.3 Nachsilbenanalysatoren für einfache Präzedenzsprachen

Der Abschnitt 3.3.3 gliedert sich in zwei Abschnitte. Im ersten Abschnitt werden die Grundideen beschrieben, die zur Realisierung der Nachsilbenanalysatoren führen. Ein interessanter Punkt dabei ist, daß diese Ideen entstanden sind im Zusammenhang mit dem Versuch, das Verfahren von Rhodes [Rho,73] zu verbessern. Erst zu einem späteren Zeitpunkt wurde erkannt, daß es sich bei dem Verfahren um eine Realisierung der von P. Gonser vorgeschlagenen Nachsilbenanalyse handelt. Im zweiten Abschnitt werden die Nachsilbenanalysatoren in der Form von Abbildungen realisiert und Hinweise auf Möglichkeiten der Implementierung gegeben.

3.3.3.1 Anschauliche Beschreibung der Konstruktion der Analysatoren

Der naheliegende Weg zur Konstruktion von Nachsilbenanalysatoren ist der folgende: Zunächst berechnet man in Abhängigkeit der Grammatik der zugrunde liegenden Sprache für jedes terminale Zeichen a eine Grammatik für die Nachsilbensprache $NS.a$. Die dabei auftretenden Grammatiken definieren dann Analysatoren für diese Sprachen. Dieser Weg ist für Anwendungen in der Praxis jedoch nicht brauchbar, da er zu aufwendig ist. Deswegen wollen wir versuchen die Nachsilbenanalysatoren in einer Weise zu konstruieren, die es uns erlaubt, den ursprünglichen Präzedenzanalysealgorithmus, wie die Präzedenzrelationen der Ausgangsgrammatik zur Analyse von Nachsilben zu verwenden. Der Ausgangspunkt

für diese Konstruktion ist die folgende Überlegung:

Analysatoren für einfache Präzedenzsprachen benötigen zur Bestimmung des Endes der Primphrase der aktuellen Satzform nur die Präcedenzrelationen zwischen den Zeichen der Satzform. Die Satzform selbst, bestehend aus dem schon erstellten Kellerinhalt und dem noch zu analysierenden Teil der Eingabezeichenfolge spielt dabei keine Rolle. Deshalb kann man auf eine Zeichenfolge $u := a_j \dots a_k \in \Sigma^*$, von welcher festgestellt werden soll, ob sie ein Element der Nachsilbensprache $NS.a_j$ ist, den zugrunde liegenden Präcedenzanalysealgorithmus anwenden. Dieser analysiert u solange, bis entweder ein Fehler erkannt, d.h. ein Fehlerausgang⁽¹⁾ erreicht wird, oder ein Ende einer Primphrase gefunden (Zeichen, welches mit dem folgenden Zeichen die Relation $\triangleright$ hat) wird, zu dem kein Anfang im Keller steht (Zeichen, welches mit seinem Vorgänger die Relation $\triangleleft$ hat).

Im zuletzt genannten Fall tritt eine der drei folgenden Situationen ein:

1. Der Kellerinhalt ist kein Suffix einer rechten Seite einer Produktion (der zu Grunde liegenden Grammatik).
2. Der Kellerinhalt ist Suffix von genau einer rechten Seite einer Produktion.
3. Der Kellerinhalt ist Suffix von mehr als einer rechten Seite einer Produktion.

Im Fall 1 ist u kein Element von $NS.a_j$. In den Fällen 2 und 3 ist der bisher analysierte Teil von u ein Infix eines Wortes $vu \in L$, und der dadurch entstandene Kellerinhalt ein Suffix einer Primphrase der zugrunde liegenden Grammatik. Deshalb bezeichnen wir diesen Kellerinhalt als potentielles Primphasensuffix, oder abgekürzt als PPS.

Beispiel 3.10: Potentielle Primphasensuffixe

1. Sei u die durch

$u := \text{while id then id} := \text{id end.}$

⁽¹⁾ Zunächst sind nur die Fehlerausgänge von A_{F2} denkbar, d.h. es werden Zeichenpaarfehler, Reduktionsfehler und Kellerungsfehler erkannt.

definierte Zeichenfolge, u ist kein Suffix von L_1 . Bei der Analyse von u ergibt sich die Zeichenfolge

while E then

als potentielltes Primphrasensuffix. Da die Zeichenfolge *while E then* kein Suffix einer rechten Seite einer Produktion von G_1 ist, ist sie insbesondere auch kein Suffix einer Primphrase.

2. Sei u das durch

$u := + id * id \quad end.$

definierte Suffix von L_1 . Bei der Analyse von u ergibt sich $+ F$ als potentielltes Primphrasensuffix, da $F \triangleright *$ gilt. Ferner ist $+ F$ Suffix der Primphrase $T + F$, welche bei der Analyse der Zeichenfolge

program id;

begin

$id := id + id * id$

end.

auftritt.

3. Sei u das durch

$u := id ; id := id \quad end.$

definierte Suffix von L_1 .

Wegen $id \triangleright ;$ ist id das zuerst auftretende potentielle Primphrasensuffix. In diesem Fall ist nicht eindeutig bestimmt, zu welcher der Primphrasen:

program id ($S \rightarrow \text{program id}$)

id ($F \rightarrow id$)

var Il1 id ($Vh \rightarrow \text{var Il1 id}$)

id Suffix ist.

Auf der eben geschilderten Überlegung beruht der Vorwärtsschritt der Fehlerbehandlungsmethode von Rhodes (siehe Abschnitt 3.2.3). Bei Rhodes wird nach dem Erkennen eines Fehlers solange weiter analysiert, bis entweder erneut ein Fehler entdeckt, oder ein potentielltes Primphrasensuffix gefunden wird. Im ersten Fall ist die von den beiden Fehlern eingeschlossene Zeichenfolge, im zweiten Fall das potentielle Primphrasensuffix der erste Kandidat für eine Korrektur des Kellers.

Ein erster Versuch ⁽¹⁾ der Konstruktion eines Nachsilbenanalysators aus

⁽¹⁾ Den entsprechenden Algorithmus bezeichnen wir im folgenden mit V1.

dem gegebenen Analysator besteht darin, im Fall eines gefundenen potentiellen Primphrasensuffixes zunächst zu prüfen, ob das potentielle Primphrasensuffix ein Suffix einer rechten Seite ist. Ist dies nicht der Fall, so ist ein Fehler in der Nachsilbe entdeckt und die Nachsilbenanalyse ist beendet. Ist dies der Fall, so wird das potentielle Primphrasensuffix vergessen und die Analyse mit dem nächsten Zeichen fortgesetzt, bis entweder ein weiterer Fehler oder ein weiteres potentielles Primphrasensuffix entdeckt ist. Dieser Algorithmus wird solange wiederholt bis die Zeichenfolge vollständig analysiert ist.

Beispiel 3.11: Ein erster Versuch der Realisierung der Nachsilbenanalyse

Die Zeichenfolge u , definiert durch

```

u := + id * id ;
    id [id] ;
    while id + id * id ;
    end.

```

ist keine Nachsilbe von L1. Dies wird vom Algorithmus V1 jedoch nicht erkannt. Bei Analyse von u mit Hilfe von V1 ergeben sich die potentiellen Primphrasensuffixe:

$+ F$	durch Analyse von	$+ id$
$* F$	durch Analyse von	$* id$
$id [E]$	durch Analyse von	$id [id]$
$;$	durch Analyse von	$;$
$while E$	durch Analyse von	$while id + id * id$
$;$	durch Analyse von	$;$
end	durch Analyse von	end
$.$	durch Analyse von	$.$

Alle potentiellen Primphrasensuffixe sind Suffixe von rechten Seiten von Produktionen, das heißt V1 erkennt keinen Fehler in u . u enthält jedoch Fehler, da eine der anstatt u beabsichtigten Zeichenfolgen $\tilde{u}$ durch

```

u :=      + id * id ;
      id := id [id] ;
      while id + id * id  do id := id ;
      end.

```

definiert ist.

Δ

Die Ursache für die schlechte Fehlererkennung von V1 liegt darin, daß durch das wiederholte Vergessen von potentiellen Primphrasensuffixen nicht mehr entschieden werden kann, ob das potentielle Primphrasensuffix aufgrund eines Fehlers in der zu analysierenden Zeichenfolge u entstanden ist, oder ob es sich um eine Konsequenz des gelöschten Kellers handelt. Das obige Problem wird vermieden, wenn man den Algorithmus V1 wie folgt zu einem Algorithmus V2 erweitert.

Ist bei der Analyse von u mit Hilfe von V1 ein Suffix λ einer Primphrase gefunden und ist a das aktuelle Zeichen, so bestimmt man zunächst die Menge $LS(\lambda, a)$, definiert durch:

$$LS(\lambda, a) := \{A \in N \mid \exists A \rightarrow \lambda \lambda' \in P : \lambda' \in \Gamma^* \wedge A \in a\}$$

Jedes Element von $LS(\lambda, a)$ ist geeignet λ zu ersetzen und dadurch die Fortsetzung der Analyse zu ermöglichen.

Enthält nämlich $LS(\lambda, a)$ genau ein Element A und ist die Zeichenfolge u eine Nachsilbe von L, so existiert ein $v \in \Sigma^*$ mit $vw \in L$, für dessen Rechtsableitung

$$S \xrightarrow{R^*} \alpha \lambda' w \xrightarrow{R^*} v u \quad \text{mit } w \in \Sigma^*, \alpha \in \Gamma^*$$

gilt, daß $\lambda' \lambda$ Primphrase von $\alpha \lambda' \lambda w$ ist, welche auf A reduziert wird.

Enthält $LS(\lambda, a)$ mehr als ein Element, und ist u ein Suffix von L, so existiert mehr als ein Element A in $LS(\lambda, a)$ für das die obige Aussage gilt.

Enthält also $LS(\lambda, a)$ genau ein Element, so wird λ durch dieses ersetzt und die Analyse fortgesetzt.

Enthält $LS(\lambda, a)$ mehr als ein Element, so ersetzt man λ durch eines der Elemente von $LS(\lambda, a)$. Da erst im weiteren Verlauf der Analyse festge-

stellt werden kann, ob die richtige Ersetzung gewählt wurde, müssen vor der Fortsetzung der Analyse der Zeiger auf das Eingabemedium sowie die bereits versuchten Ersetzungen gemerkt werden, so daß beim Auftreten eines Fehlers auch die anderen Elemente von $LS(\lambda, a_k)$ eingesetzt werden können. Dabei ist zu beachten, daß dieses Problem rekursiv auftreten kann. Im nächsten Abschnitt wird bewiesen, daß der eben beschriebene Algorithmus V2 einen Nachsilbenanalysator für einfache Präzedenzsprachen darstellt. In diesem Abschnitt wird die Arbeitsweise von V2 an einigen Beispielen demonstriert. Ferner wird an Beispielen gezeigt, daß V2 in der Nachsilbe i.allg. nicht frühestmöglich erkennt und wie dies behoben werden kann.

Beispiel 3.12: Der Algorithmus V2

1. Sei u das durch

$u := + id * id \text{ end.}$

definierte Suffix von $L1$.

Die Analyse von u mit Hilfe von V2 wird durch das folgende Schema beschrieben.

Element aus $LS(\lambda, a)$	anal.Zei- chenfolge	aktuelles Zeichen a	Keller- inhalt λ	$LS(\lambda, a)$	Fehler- situation
	$+ id$	$*$	$+ F$	T	
T	$* id$	end	$T * F$	T	
T	$-$	end	T	$T1$	
$T1$	$-$	end	$T1$	E	
E	$-$	end	E	A, W	
W	$-$	end	W	$-$	W hat keine Re- lation zu end
A	$-$	end	A	Cs	
Cs	$-$	end	Cs	St, Ic	
St	$-$	end	St	L	
L	$-$	end	L	$L1$	
$L1$	end	$.$	$L1 \text{ end}$	C	
C	$.$	$*$	$C.$	Ch	
Ch	$-$	$\#$	Ch	B	
B	$-$	$\#$	B	S	

Das heißt u wird auf S reduziert.

2. Sei die Zeichenfolge u definiert durch

```

u := + id * id ;
      id [ id ]
      end.

```

Dann ist u kein Suffix von $L1$. Das folgende Schema beweist, daß $V2$ dies erkennt.

Element aus $LS(\lambda, a)$	anal. Zeichen- folge	aktuelles Zeichen a	Keller- inhalt λ	$LS(\lambda, a)$	Fehler- situation
	$+ id$	$*$	$+ F$	T	
T	$* id$	$;$	$T + F$	T	
T	$-$	$;$	T	$T1$	
$T1$	$-$	$;$	$T1$	E	
E	$-$	$;$	E	A, W	
W	$-$	$;$	$-$	$-$	W hat keine Relation zu $;$
A	$-$	$;$	A	Cs	
Cs	$-$	$;$	Cs	St, Ic	
St	$-$	$;$	St	L	
L	$;$	id	$L;$	$L2$	
$L2$	$id [id]$	end	$L2 id[E]$	$-$	Reduktionsfeh- ler, da $id [E]$ keine rechte Seite ist.

Δ

Die obigen Beispiele zeigen die Arbeitsweise von $V2$ für den Fall, in dem nicht zurückgesetzt werden muß. Das folgende Beispiel zeigt, daß der Fall, in dem $V2$ zurücksetzen muß, sogar in der Sprache $L1$ vorkommt, d.h. erst recht in den in der Praxis verwendeten Programmiersprachen.

Beispiel 3.13: Rekursives Auftreten von $|LS(\lambda, a)| > 1$

Sei das Suffix u von $L1$, definiert durch

```
u := id ;
    begin id := id end.
```

gegeben.

Bei der Analyse von u ist das erste PPS id . Da $LS(id, ;) = \{Pi, Vh, F\}$ gibt es drei Möglichkeiten für Ersetzungen von id . Die Ersetzung von id durch F führt zu einer Fehlermeldung beim Lesen des Zeichens . . Das heißt, erst nachdem die gesamte Zeichenfolge analysiert ist, wird erkannt, daß F die falsche Alternative war. Ferner tritt nach dem Ersetzen von id durch F während der Analyse von $begin\ id := id\ end.$ der Fall, daß $|LS(\lambda, a)| > 1$ ist noch einmal, d.h. insgesamt rekursiv auf, denn wie im vorigen Beispiel wird F auf CS reduziert und es gilt: $LS(CS, ;) = \{St, IC\}$. Die Ersetzungen id durch Vh bzw. id durch Pi führen zu keinen Fehlermeldungen, jedoch aber zu verschiedenen Analysen von u , was bedeutet, daß es zwei Möglichkeiten der Analyse von u gibt.

Δ

Das nächste Beispiel beschäftigt sich mit der Fehlererkennung von V2. Da V2 nur Zeichenpaarfehler, Kellerungsfehler und Reduktionsfehler erkennt, ist es nach den Ausführungen des Abschnitts 2 offensichtlich, daß V2 i.allg. Fehler nicht frühestmöglich meldet.

Beispiel 3.14: Zur Fehlererkennung von V2

1. Bei der Analyse einer Zeichenfolge

```
u := while id = id then id := id ; ...
```

wird wegen der fehlenden Prüfung auf RS-Fehler erst beim Lesen von id (nach $then$) ein Fehler erkannt.

2. Bei der Analyse einer Zeichenfolge

```
u := [ id := id + id ] := id ; ...
```

wird erst nach dem Lesen von $\bar{J}$ und der damit verbundenen Reduktion von $\bar{id} := id + id$ auf A ein Fehler erkannt, weil keine Mengen von gesuchten Variablen berechnet werden.

△

Um die in den obigen Beispielen aufgeführten Fehlertypen rechtzeitig zu erkennen, muß der Algorithmus V2 um die Prüfung auf rechte Seiten, wie die Berechnung der gesuchten Variablen und der darauf beruhenden Fehlererkennung erweitert werden. Damit ist dann insbesondere das Problem der Konstruktion eines Nachsilbenanalysators mit LP-Eigenschaft gelöst.

Die in diesem Abschnitt anschaulich entwickelten Algorithmen werden im nächsten Abschnitt formal definiert. Ferner werden dort für die hier behaupteten Eigenschaften Beweise geführt.

3.3.3.2 Formale Beschreibung der Analysatoren für Nachsilben

In diesem Abschnitt erweitern wir die im Abschnitt 2 definierten Analysefunktionen $F_1, \dots, F_4$ für einfache Präzedenzsprachen zu Analysefunktionen $\bar{F}_1, \dots, \bar{F}_4$ für die Nachsilbensprachen von einfachen Präzedenzsprachen.

Die von diesen Analysefunktionen induzierten Analysatoren $NSA_{\bar{F}_1}, \dots, NSA_{\bar{F}_4}$ sind dann Nachsilbenanalysatoren. Der von F_4 induzierte Analysator $NSA_{\bar{F}_4}$ erkennt Fehler frühestmöglich. Mit dem Analysator $\mathcal{U}_{A_{F_4}, NSA_{\bar{F}_4}}$ ist das Problem der Konstruktion von minimalen Nachsilbenanalysen für einfache Präzedenzsprachen gelöst. Gleichzeitig sind mit $\mathcal{U}_{A_{F_2}, NSA_{\bar{F}_2}}$ und $\mathcal{U}_{A_{F_3}, NSA_{\bar{F}_3}}$ (von A_{F_2} und $NSA_{\bar{F}_2}$, bzw. A_{F_3} und $NSA_{\bar{F}_3}$ induzierte Analysatoren) weniger aufwendig zu implementierende fehlerbehandelnde Analysealgorithmen gewonnen, welche den bisher bekannten Fehlerbehandlungsalgorithmus dadurch überlegen sind, daß sie keine Folgefehler produzieren.

3.3.3.2.1 Analysefunktionen für Nachsilbensprachen

Unser Begriff der Analysefunktion für Nachsilbensprachen ist eine Erweiterung des Begriffs der Analysefunktion für einfache Präzedenzsprachen.

Definition 3.12

Sei $G = (N, \Sigma, P, S)$ eine EPG, Konf_G die Menge der Konfigurationen über G , dann gilt:

Eine Abbildung $F : \text{Konf}_G \rightarrow 2^{\text{Konf}_G}$ heißt eine Analysefunktion für die Nachsilbensprachen von $L = L(G)$, oder eine Nachsilbenanalysefunktion für $L = L(G)$ genau dann, wenn für $u = a_1 \dots a_n \in \Sigma^*$ gilt:

1. $u \in \text{NS.L}$ $\bigwedge \exists$ eine Folge von Konfigurationen $k_i \in \text{Konf}_G$, $i = 0, \dots, p$ mit den Eigenschaften:
 - 1.1 $k_0 = (\#, u\#)$
 - 1.2 $k_{i+1} \in F(k_i)$ für $i = 0, \dots, p-1$
 - 1.3 $k_p = \text{true}$
2. $u \notin \text{NS.L}$ $\bigvee \exists$ eine Folge $k_i \in \text{Konf}_G$, $i = 0, \dots, p$ mit den Eigenschaften:
 - 2.1 $k_0 = (\#, u\#)$
 - 2.2 $k_{i+1} \in F(k_i)$ für $i = 0, \dots, p-1$
 - 2.3 $k_p \in N$ mit: $0 \leq k_p \leq n-q$, wobei $a_1 \dots a_q$ das längste Präfix von u bzgl. NS.L ist.

Definition 3.13

Sei $G = (N, \Sigma, P, S)$ eine EPG, F eine Nachsilbenanalysefunktion für $L=L(G)$, $u \in \Sigma^+$, dann gilt:

1. Sei $u \in \text{NS.L}$, dann heißt jede Folge $k_0, \dots, k_p$ von Konfigurationen mit den in Definition 3.12,1 definierten Eigenschaften eine Analyse von u , mit Hilfe von F .
2. Sei $u \notin \text{NS.L}$ dann heißt jede Folge $k_0, \dots, k_p$ von Konfigurationen mit den in Definition 3.12,2 definierten Eigenschaften und minimalem

k_p eine Analyse von u, mit Hilfe von F.

3. Jede Konfiguration k_j einer Analyse von u heit eine mit Hilfe von F definierte Konfiguration von u.

Jede Nachsilbenanalysefunktion F fr L definiert auf natrlich Weise einen Analysator fr L.

Satz 3.6

Sei $G = (N, \Sigma, P, S)$ eine EPG, F eine Nachsilbenanalysefunktion fr $L = L(G)$, dann gilt:

Die Abbildung

$$NSA_F : \Sigma^+ \rightarrow \text{Ekonf}$$

definiert durch :

$$NSA_F(u) := \begin{cases} \text{true} & \text{falls die Relation } (\#, u\#) F^+ \text{ true} \\ & \text{gilt} \\ |u| + 1 - k_p & \text{falls die Relation } (\#, u\#) F^+ \text{ true} \\ & \text{nicht gilt, und } k_0, \dots, k_p \text{ eine} \\ & \text{Analyse von u ist.} \end{cases}^1$$

ist ein Nachsilbenanalysator fr L.

Beweis:

Sei $u \in \Sigma^+$ und $k_0, \dots, k_p$ eine Analyse von u mit Hilfe von F, dann gilt:

1. Falls $u \in NS.L$ ist, gilt $k_p = \text{true}$
2. Falls $u \notin NS.L$ ist, gilt:
 - 2.1 $k_p = 0$ falls $u \in \text{Prfix}(NS.L)$
 - 2.2 $0 < k_p \leq n$ falls $u \notin \text{Prfix}(NS.L)$

Daraus folgt die Behauptung auf triviale Weise.

q.e.d.

Die Definition der Nachsilbenanalysefunktionen unterscheidet sich von der Analysefunktion nur dadurch, da die von Nachsilbenanalysefunktionen

(1) F kann aufgefat werden als Relation ber Konf_G , d.h. F^+ ist die transitive Hlle von F.

erzeugten Analysen in der Regel nicht eindeutig sind, nämlich genau dann, wenn die entsprechenden Nachsilbensprachen zweideutig sind. Deshalb können die in Abschnitt 2 entwickelten Analysefunktion $F_1, \dots, F_4$ zu Nachsilbenanalysefunktionen $\overline{F}_1, \dots, \overline{F}_4$ erweitert werden. Der induzierte Analysator $NSA_{\overline{F}_2}$ entspricht dem im vorigen Kapitel entwickelten Algorithmus V2. Deshalb ist der von A_{F_2} und $NSA_{\overline{F}_2}$ induzierte Analysator $\mathcal{M}_{A_{F_2}, NSA_{\overline{F}_2}}$ ein fehlerbehandelnder Analysealgorithmus, welcher für Anwendungen in der Praxis brauchbar ist. Die induzierten Analysatoren $NSA_{\overline{F}_3}$ und $NSA_{\overline{F}_4}$ realisieren die für V2 vorgeschlagenen Erweiterungen, insbesondere hat $NSA_{\overline{F}_4}$ die LP-Eigenschaft.

3.3.3.2.2 Erweiterung der Analysefunktionen für einfache Präzedenzsprachen zu Analysefunktionen für deren Nachsilbensprachen

Die in Abschnitt 3.3.3.1 beschriebenen Analysatoren für Nachsilbensprachen von einfachen Präzedenzsprachen unterscheiden sich von den Analysatoren für einfache Präzedenzsprachen durch die Möglichkeit der Reduktion von potentiellen Primphrasensuffixen und durch die Notwendigkeit, mit Zurücksetzen zu arbeiten. Deswegen werden die Analysefunktionen F_1 bis F_4 dadurch zu Analysefunktionen $\overline{F}_1$ bis $\overline{F}_4$ für Nachsilbensprachen erweitert, so daß Reduktionen für potentielle Primphrasensuffixe zugelassen werden und daß in diesem Fall das Bild einer Konfiguration nicht mehr eindeutig bestimmt ist, d.h. die Analysefunktionen nicht mehr deterministisch sind. Ferner muß bei der Definition von F_3 die Prüfung auf RS-Fehler sowie bei der Definition von F_4 die Berechnung der Mengen von gesuchten Variablen den durch das Auftreten von unvollständigen rechten Seiten neu hinzukommenden Analysesituationen angepaßt werden.

Satz 3.7

Sei $G = (N, \Sigma, P, S)$ eine EPG, $L = L(G)$, $Konf_G$ die Menge der Konfigurationen über G , dann gilt:

Die Abbildung $\overline{F}_1 : Konf_G \rightarrow 2^{Konf_G}$, definiert durch:

1. $\overline{Fl}(\#a, u\#) := (\#a, u\#)$ falls $u \in \Sigma^+$ und $u = au^{(1)}$, $a \in \Sigma$, $u' \in \Sigma^*$
 $(\#A, u\#)$ (2) falls (1)
 1. $X_i \succ \text{Anf}(u\#)$
 2. $X_m \doteq X_{m+1}$ für $m=1, \dots, i-1$
 3. $\exists A \rightarrow \beta X_1 \dots X_i \in P$
2. $\overline{Fl}(\#\alpha, u\#) := \begin{cases} |u\#|-1 & \text{falls} \\ \overline{Fl}(\#\alpha, u\#) & \text{sonst (Fälle (1)-(6) aus Satz 1.2)} \end{cases}$ (2)
 1. $X_i \succ \text{Anf}(u\#)$
 2. $X_m \doteq X_{m+1}$ für $m=1, \dots, i-1$
 3. $\nexists A \rightarrow \beta X_1 \dots X_i \in P$

wo $\alpha = X_0 X_1 \dots X_i$, $X_j \in \Gamma$, $u \in \Sigma^+$ gilt,

ist eine Nachsilbenanalysefunktion für L.

Bemerkung 3.3

1. Im Fall (1) der obigen Definition sagen wir, die Konfiguration $(\#A, u\#)$ entsteht aus der Konfiguration $(\#\alpha, u\#)$ durch eine Reduktion einer unvollständigen rechten Seite, wenn für alle Produktionen $A \rightarrow \beta X_1 \dots X_i \in P$, $\beta \neq \epsilon$ gilt.

2. Den in (2) erkannten Fehlertyp bezeichnen wir mit Reduktionsfehler.

Beweis:

Zu zeigen ist:

Sei $u = a_1 \dots a_n \in \Sigma^+$ dann gilt:

1. $u \in \text{NS}.a_1 \iff \exists k_i \in \text{Konf}_G \quad i = 0, \dots, p \text{ mit:}$

$$1. k_0 = (\#\alpha, u\#)$$

(1) Für alle weiteren Definitionen und Sätze dieser Arbeit gelte o.B.d.A. $\text{NS}.a \neq \emptyset \quad \forall a \in \Sigma$

(2) Die Bilder von $\overline{Fl}$ sind Elemente von 2^{Konf_G} , also Mengen. In der obigen Definition (wie auch in allen weiteren Definitionen) werden die Mengenklammern weggelassen.

2. $k_{i+1} = \overline{Fl}(k_i)$ für $i = 0, \dots, p-1$
 3. $k_p = \text{true}$
2. $u \notin \text{NS}.a_1 \} \exists k_i \in \text{Konf}_G, i = 0, \dots, p$ mit:
1. $k_0 = (\#, u\#)$
 2. $k_{i+1} = \overline{Fl}(k_i)$ für $i = 0, \dots, p-1$
 3. $k_p \in \mathbb{N}$ mit: $0 \leq k_p \leq n-q$ wobei $a_1 \dots a_q$ das längste Präfix von u bzgl. $\text{NS}.a_1$ ist.

Beweis von 1

" $\}$ "

Sei $u \in \text{NS}.a_1$, dann existiert ein $v = b_1 \dots b_1 \in \Sigma^*$ mit $vu \in L$.

Zu vu existiert nach Satz 1.2 eine Folge $\bar{k}_i \in \text{Konf}_G, i = 0, \dots, q$ mit den Eigenschaften:

1. $\bar{k}_0 = (\#, vu\#)$
2. $\bar{k}_{j+1} = Fl(\bar{k}_j)$ für $j = 0, \dots, q-1$
3. $\bar{k}_{q-1} = (\#S, \#)$ $\bar{k}_q = Fl(\bar{k}_{q-1}) = \text{true}$

Die gesuchte Folge k_i definieren wir durch vollständige Induktion:

Induktionsanf.: $i = 1$

Sei $u\# = a_1 u'$ und sei $j_0 \in \mathbb{N}$ so gewählt, daß

$$\bar{k}_{j_0+1} = (\#\bar{a}a_1, u')$$

gilt, dann definieren wir k_1 durch:

$$k_1 := (\#a_1, u').$$

Induktionsschritt:

Sei für $j \in \mathbb{N}$ mit $j_0 + j < q-1$, die Folge $k_i, i = 1, \dots, j$ definiert, so daß

$$k_j = (\#\alpha, \tilde{u}\#)$$

und

$$\bar{k}_{j_0+j} = (\#\bar{a}\alpha, \tilde{u}\#) \text{ gilt}$$

, dann definieren wir k_{j+1} durch:

$$k_{j+1} := \left\{ \begin{array}{ll} (\# \alpha a, u') & \text{falls} \\ & 1. \tilde{u} \# = au', a \in \Sigma, u' \in \Sigma^* \# \\ & 2. \text{End}(\alpha) \dot{=} a \text{ oder } \text{End}(\alpha) \leq a \\ \\ (\# \bar{\alpha} A, \tilde{u} \#) & \text{falls} \\ & 1. \text{End}(\alpha) \triangleright \text{Anf}(\tilde{u} \#) \\ & 2. \text{Für } \bar{k}_{j_0+j-1} = \text{Fl}(\bar{k}_{j_0+j}) \text{ gilt:} \\ & \quad \bar{k}_{j_0+j+1} = (\# \bar{\alpha} \bar{\alpha} A, \tilde{u} \#) \text{ mit } A \rightarrow \psi \in P \\ & \quad \quad \quad \wedge \alpha = \bar{\alpha} \psi \\ \\ (\# A, \tilde{u} \#) & \text{falls} \\ & 1. \text{End}(\alpha) \triangleright \text{Anf}(u \#) \\ & 2. \text{Für } \bar{k}_{j_0+j-1} = \text{Fl}(\bar{k}_{j_0+j}) \text{ gilt:} \\ & \quad \bar{k}_{j_0+j+1} = (\# \bar{\alpha} A, \tilde{u} \#) \text{ mit } A \rightarrow \psi \in P \\ & \quad \quad \quad \wedge \bar{\alpha} \alpha = \bar{\alpha} \psi \\ & \quad \quad \quad \wedge \psi = \beta \alpha, \beta \neq \epsilon \end{array} \right.$$

Definiert man ferner k_0, k_p durch

$$k_0 := (\#, u \#)$$

und

$$k_p := \text{true} \quad \text{für } p := q - j_0$$

dann hat die Folge $k_i, i = 0, \dots, p$ die gewünschten Eigenschaften.

Daraus folgt die Behauptung.

q.e.d. (" } ")

" { "

Sei eine Folge von Konfigurationen $k_i, i = 0, \dots, q$ mit den Eigenschaften 1 - 3 gegeben. Dann haben wir ein $v \in \Sigma^*$ zu konstruieren mit $vu \in L$. Dies geschieht durch vollständige Induktion über die Anzahl j der Konfigurationen k_i , die durch Reduktionen von unvollständigen rechten Seiten entstanden sind.

Induktionsanfang: $k = 0$

Für alle Konfigurationen k_i , $i=0, \dots, q-1$ gilt, nach Definition von $\overline{Fl}$

$$\overline{Fl}(k_i) = Fl(k_i)$$

d.h. aber, daß für die Konfigurationen k_i

1. $k_0 = (\#, u\#)$
2. $k_{i+1} = Fl(k_i)$ für $i = 0, \dots, q-1$
3. $k_q = \text{true}$

gilt.

Aus Satz 1.2 folgt, daß $u \in L$ gilt, d.h. daß $v := \varepsilon$ die zu konstruierende Zeichenfolge ist.

Induktionsschritt:

Sei die Behauptung für $0 < k < m$ gültig, dann ist sie für $k = m + 1$ zu zeigen.

Sei i_0 der kleinste Index mit der Eigenschaft k_{i_0+1} entsteht aus k_{i_0} durch die Reduktion einer unvollständigen rechten Seite. Dann gilt

1. $k_{i_0} = (\# \alpha, u\#)$
2. $k_{i_0+1} = (\# A, u\#)$
3. $\forall A \rightarrow \beta \alpha \in P$ gilt $\beta \neq \varepsilon$

Sei $\bar{v} \in \Sigma$ eine terminale Zeichenfolge, welche aus β abgeleitet werden kann, d.h. für die

$$\beta \xrightarrow{R}^* \bar{v}$$

existiert. Dann existiert eine Folge von Konfigurationen $\bar{k}_i$, $i=0, \dots, \bar{q}$ mit den Eigenschaften:

1. $\bar{k}_0 := (\#, \bar{v}u\#)$
2. $\bar{k}_{i+1} = \overline{Fl}(\bar{k}_i)$ $i = 0, \dots, \bar{q}-1$
3. $\bar{k}_{\bar{q}} = \text{true}$

4. $\exists r_0 \in \{1, \dots, \bar{q}-1\} : k_{r_0} = (\#A, u\#)$
5. Genau m der Konfigurationen $\bar{k}_i$ sind durch Reduktionen von unvollständigen rechten Seiten entstanden.

Die Existenz einer Folge von Konfigurationen mit den Eigenschaften 1-4 folgt aus der Wahl von $\bar{v}$ mit $\beta \xrightarrow{R^*} \bar{v}$.

Die Eigenschaft 5 gilt, weil nach Konstruktion keine der Konfigurationen $\bar{k}_i$ mit $0 < i < r_0 + 1$ durch Reduktionen von unvollständigen rechten Seiten entsteht. Ferner gilt $\bar{k}_{r_0+k} = k_{i_0+k}$ für $1 < k < q - r_0$ und nach Voraussetzung sind genau m der Konfigurationen $\bar{k}_{r_0+k}$ mit $1 < k < q - r_0$ durch Reduktionen von unvollständigen rechten Seiten entstanden.

D.h. auf die Folge $\bar{k}_i$ $i = 0, \dots, \bar{q}$ ist die Induktionsvoraussetzung anwendbar, und daraus folgt die Behauptung.

q.e.d. (" $\angle$ ")
q.e.d. (1)

Beweis von 2:

Der Beweis verläuft vollständig analog der Richtung " $\rangle$ " von 1.

q.e.d. (2)

q.e.d. (Satz 3.8)

Der von $F1$ induzierte Nachsilbenanalysator NSA_{F1} definiert zusammen mit A_{F1} nach Satz 3.2 den fehlerbehandelnden Analysator $\mathcal{U}_{A_{F1}, NSA_{F1}}$, der die Eigenschaft E1 nur mangelhaft erfüllt, aber dennoch für Anwendungen in der Praxis brauchbar ist.

Beispiel 3.15: Analyse mit Hilfe von $\mathcal{U}_{A_{F1}, NSA_{F1}}$

Sei die bezüglich L1 fehlerhafte Zeichenfolge u definiert durch

```

u := program id ;
    begin
        id := id [ ( id + id )↑F1 + id * id ;
        id ;↑F2
        while id then id := id ;↑F3
        if id := id id then id = id↑F4
        if id↑F5
        end
    end.
    ↑F7

```

aus Beispiel 3.8 gegeben. Wie bereits erwähnt, enthält sie sieben Fehler, welche mit $\uparrow F_k$, $k = 1, \dots, 7$ markiert sind. Die Analyse von u mit Hilfe von $\mathcal{U}_{A_{F1}, NSA_{F1}}$ wird durch die folgende Tabelle wiedergegeben.

Fehler	Kellerinhalt bei ⁽¹⁾ Fehlererkennung	Relation	akt. Zeichen	Fehlertyp	FA
F1	$\# \leftarrow Ph \leftarrow begin \leftarrow id \dot{=} \dot{=} G \leftarrow id \dot{=} [\leftarrow (\dot{=} E \dot{=}]$	$\triangleright$	+	RS-Fehler	<i>id</i>
F2	$\# L2?F$	$\triangleright$	;	Reduktionsfehler	<i>while</i>
F3	$\# while \dot{=} E \dot{=} then$	$\triangleright$	<i>id</i>	Reduktionsfehler	:
F4	$\# := G \leftarrow E \dot{=}$	$\triangleright$	;	RS-Fehler	<i>if</i>
F5	$\# if \leftarrow id \dot{=} \dot{=} G \dot{=} E \dot{=} then$	$\triangleright$	<i>id</i>	RS-Fehler	=

In der obigen Tabelle repräsentiert die mit "aktuelles Zeichen" bezeichnete Spalte dasjenige Zeichen, bei dem der Fehler erkannt wurde. Die mit "Relation" bezeichnete Spalte repräsentiert die Relation des obersten Kellerelementes zum aktuellen Zeichen. Die mit "FA" bezeichnete Spalte repräsentiert dasjenige Zeichen, bei dem die Analyse mit Hilfe von $\mathcal{U}_{A_{F1}, NSA_{F1}}$ fortgesetzt wird. Daraus ergibt sich die folgende Zerlegung x_i , $i = 1, \dots, 6$ der Zeichenfolge u

- $x_1 = \text{program } id ; id := id [(id + id]$
- $x_2 = id * id ; id ;$
- $x_3 = while \quad id \text{ then } id$
- $x_4 = := id) ;$
- $x_5 = if \quad id := id + id \text{ then } id$
- $x_6 = = id \text{ end end.}$

(1)

Zwischen den geklammerten Zeichen sind zur Verbesserung der Lesbarkeit die Präzedenzrelationen notiert.

Das letzte Zeichen jeder Zeichenfolge x_i ist dasjenige Zeichen, bei dem der Fehler erkannt wird. Die obige Zerlegung zeigt, daß alle Fehler (außer F2) verspätet, und die Fehler F6 und F7 überhaupt nicht erkannt werden.

Der Grund für die späte Fehlererkennung von $\mathcal{U}_{A_{F1}, NSA_{\overline{F1}}}$ ist die späte Fehlererkennung der Analysefunktion $\overline{F1}$. Wegen der verspäteten Erkennung von F5 wird auch F6 nicht erkannt.

Da sich die Analysefunktionen F1 bis F4 nur dadurch unterscheiden, daß mit zunehmendem Index eine frühere Fehlererkennung erfolgt, ist zu erwarten, daß die von den Analysefunktionen $\overline{F2}$, $\overline{F3}$ und $\overline{F4}$ induzierten Analysatoren bessere Zerlegungen konstruieren.

Satz 3.8:

Sei $G = (N, \Sigma, P, S)$ eine EPG, $L = L(G)$, Konf_G die Menge der Konfigurationen über G , dann gilt:

Die Abbildung $\overline{F2} : \text{Konf}_G \rightarrow 2^{\text{Konf}_G}$, definiert durch:

$$\begin{aligned}
 1. \overline{F2}(\# \alpha, u \#) &:= (\# \alpha, u' \#) && \text{falls } u = au', a \in \Sigma, u' \in \Sigma^* \\
 2. \overline{F2}(\# \alpha, u \#) &:= \left\{ \begin{array}{ll} |u \#| - 1 & \text{falls} \\ 1. X_i \triangleright \text{Anf}(u \#) \\ 2. \exists p \in \mathbb{N}^0 \text{ mit } p = \max_{m=0, \dots, i-1} (X_m \triangleleft X_{m+1}) \\ 3. \exists A \rightarrow X_{m+1} \dots X_i \in P \\ 4. X_m ? A \end{array} \right. && (4) \\
 &&& \overline{F1}(\# \alpha, u \#) \text{ sonst (Fälle (1)-(3) aus Satz 3.7)}
 \end{aligned}$$

wo $\# \alpha = X_0 X_1 \dots X_i$, $X_0 = \#$, $X_j \in \Gamma$ für $j = 1, \dots, i$ gilt, ist

eine Nachsilbenanalysefunktion für L .

$\overline{F2}$ unterscheidet sich von $\overline{F1}$ nur hinsichtlich des Zeitpunkts der Fehlererkennung, deshalb ist der Beweis von Satz 3.8 vollständig analog zum Beweis von Satz 3.7.

Die von der Analysefunktion $F2$ abgeleitete Analysefunktion $\overline{F2}$ unterscheidet sich von $\overline{F1}$ nur durch die Prüfung auf Kellerungsfehler. Der von A_{F2} und NSA_{F2} induzierte Analysator $\mathcal{U}_{A_{F2}, NSA_{F2}}$ unterscheidet sich deshalb nur unwesentlich von $\mathcal{U}_{A_{F1}, NSA_{F1}}$. Die Analyse der Zeichenfolge aus Beispiel 3.14 mit Hilfe von $\mathcal{U}_{A_{F2}, NSA_{F2}}$ wird bis auf $F2$ durch die Tabelle aus Beispiel 3.14 beschrieben. $F2$ wird nicht früher, sondern nur in der Form eines Kellerungs- anstatt eines Reduktionsfehlers, erkannt. $\overline{F3}$ unterscheidet sich von $\overline{F2}$ durch die Prüfung auf RS-Fehler, wobei diese auch für unvollständige rechte Seiten durchgeführt wird.

Satz 3.9

Sei $G = (N, \Sigma, P, S)$ eine EPG, $L = L(G)$, Konf_G die Menge der Konfigurationen über G , dann gilt:

Die Abbildung $\overline{F3} : \text{Konf}_G \rightarrow 2^{\text{Konf}_G}$, definiert durch:

1. $\overline{F3}(\#a, u\#) := (\#a, u'\#)$ falls $u = au'$, $a \in \Sigma$, $u' \in \Sigma^*$
2. $\overline{F3}(\#a, u\#) := \left\{ \begin{array}{ll} |u\#| - 1 & \text{falls} \\ \begin{array}{l} 1. X_i \doteq a, a = \text{Anf}(u) \\ 2. \text{Zu } q := \max_{m: \theta_m, i-1} (X_m \triangleleft X_{m+1}) \\ \quad \nexists A \rightarrow X_{q+1} \dots X_i \text{ } a \beta \in P \end{array} & (5) \end{array} \right.$
3. $\overline{F3}(\#a, u\#) := \left\{ \begin{array}{ll} |u\#| - 1 & \text{falls} \\ \begin{array}{l} 1. X_i \doteq a, a = \text{Anf}(u) \\ 2. X_m \doteq X_{m+1} \text{ für } m = 1, \dots, i-1 \\ 3. \nexists A \rightarrow \beta_1 X_1 \dots X_i \text{ } a \beta_2 \in P, \beta_1 \beta_2 \in \Gamma^*$

$$\begin{aligned}
 |u\#|-1 & \quad \text{falls} & (7) \\
 1. & X_i \triangleright \text{Anf}(u\#) \\
 2. & \exists p = \max_{m=0, \dots, i-1} (X_m \triangleleft X_{m+1}) \\
 3. & \exists A \rightarrow X_{p+1} \dots X_i \in P \\
 4. & X_p \triangleq A \\
 5. & \exists q := \max_{m=0, \dots, i-1} (X_m \triangleleft X_{m+1}) \\
 6. & \exists B \rightarrow X_q \dots X_p \beta_2 \in P, \beta_1, \beta_2 \in \Gamma^*
 \end{aligned}$$

$$\begin{aligned}
 |u\#|-1 & \quad \text{falls} & (8) \\
 1. & X_i \triangleright \text{Anf}(u\#) \\
 2. & \exists p = \max_{m=0, \dots, i-1} (X_m \triangleleft X_{m+1}) \\
 3. & \exists A \rightarrow X_{p+1} \dots X_i \in P \\
 4. & X_p \triangleq A \\
 5. & X_m \triangleq X_{m+1} \quad \text{für } m = 1, \dots, p-1 \\
 6. & \exists B \rightarrow \beta_1 X_1 \dots X_p \beta_2, \beta_1, \beta_2 \in \Gamma^*
 \end{aligned}$$

F2($\# \alpha, u\#$) sonst (Fälle (1)-(4) aus Satz 3.8)

wo $\# \alpha = X_0 \dots X_i$, $X_0 = \#$, $X_j \in \Gamma$ für $j=1, \dots, i$, $u \in \Sigma^*$ gilt,

ist eine Nachsilbenanalysefunktion für L.

$\overline{F3}$ unterscheidet sich von $\overline{F1}$ nur in der Fehlererkennung, d.h. der Beweis von Satz 3.10 ist vollständig analog zum Beweis von Satz 3.3.

Bemerkung 3.4:

1. Bei den Fällen (5) - (8) des obigen Satzes handelt es sich um die Prüfung auf RS-Fehler für die Fälle:

- 1.1 Terminales Zeichen verlängert Präfix der rechten Seite nicht
- 1.2 Terminales Zeichen verlängert Infix der rechten Seite nicht
- 1.3 Variable verlängert Präfix der rechten Seite nicht
- 1.4 Variable verlängert Infix der rechten Seite nicht.

2. Im folgenden bezeichnen wir diese RS-Fehler mit RS1 bis RS4.

Beispiel 3.16: Analyse mit Hilfe von $\mathcal{U}_{A_{F3}, NSA_{F3}}$

Die Analyse der Zeichenfolge u aus Beispiel 3.13 wird durch die folgende Tabelle wiedergegeben.

Fehler	Kellerinhalt bei Fehlererkennung	Relation	akt. Zeichen	Fehlertyp	FA
F1	* $\leftarrow Phsbegin \leftarrow id \rightarrow : \leftarrow id \rightarrow [\leftarrow (\rightarrow E$	$\rightarrow$	$]$	RS1	$+$
F2	* $L2?F$	$\rightarrow$	$;$	Kellerungsfehler	<i>while</i>
F3	* $while \rightarrow E$	$\rightarrow$	<i>then</i>	RS2	<i>id</i>
F4	* $id \rightarrow : \rightarrow G \leftarrow E$	$\rightarrow$	$)$	RS1	$;$
F5	* $L2 \leftarrow if \leftarrow id \rightarrow : \rightarrow G \rightarrow E$	$\rightarrow$	<i>then</i>	RS1	<i>id</i>

Die durch $\mathcal{U}_{A_{F3}, NSA_{F3}}$ erzeugte Zerlegung x_i , $i = 1, \dots, 6$ von u ist durch:

- $$\begin{aligned}
 x_1 &= \text{program } id ; \quad id := id [(id + id] \\
 x_2 &= + id * id ; \quad id ; \\
 x_3 &= \text{while } id \text{ then} \\
 x_4 &= id := id) \\
 x_5 &= ; \text{ if } id := id + id \text{ then} \\
 x_6 &= id = id \quad \text{end} \quad \text{end.}
 \end{aligned}$$

gegeben. Durch die Prüfung auf RS-Fehler werden die Fehler F1 - F4

frühestmöglich erkannt. F5 hingegen wird immer noch zu spät erkannt. Das hat zur Folge, daß F6 nicht erkannt wird.

Δ

F5 wird deshalb spät erkannt, weil nur geprüft wird, ob die Zeichenfolge $id : GE$ ein Anfang einer rechten Seite ist, nicht aber, ob $id : GE$ ein Anfang einer Zeichenfolge ist, die aus E ableitbar ist. Bei der Konstruktion von F4 wurden diese und weitere Prüfungen auf Fehler analoger Art durch die Einführung der Mengen von gesuchten Variablen ermöglicht. Deshalb ergibt sich $\overline{F4}$ aus $\overline{F3}$ durch Einführung von analogen Prüfungen. Dazu muß zunächst der Begriff der Mengen von gesuchten Variablen für diejenigen Kellerinhalte definiert werden, die bei der Analyse von Nachsilben auftreten können. Dies sind genau die Suffixe der BU-Präfixe der entsprechenden Grammatik.

Definition 3.14

Sei $G = (N, \Sigma, P, S)$ eine KFG, dann heißt $\gamma \in \Gamma^*$ ein Bottom-Up Suffix (BU-Suffix) von G , wenn γ ein Suffix eines BU-Präfixes von G ist.

Satz 3.10

Sei $G = (N, \Sigma, P, S)$ eine EPG, $\gamma \in \Gamma^*$ ein BU-Suffix von G , dann gilt:
Es existiert eine Folge $\psi_i \in \Gamma^*$, $i = 0, \dots, k$ mit den Eigenschaften:

1. $\psi_0 \psi_1 \dots \psi_k = \# \gamma$
2. $\psi_0 = \#$
3. Für $i = 1, \dots, k$ gilt:
 $\psi_i = x_1^i \dots x_{h_i}^i$ mit $x_q^i \in \Gamma$ für $q = 1, \dots, h_i$
 und $x_q^i \triangleq x_{q+1}^i$ für $q = 1, \dots, h_i - 1$
4. $x_{h_i}^i \triangleleft x_1^{i+1}$ für $i = 1, \dots, k-1$

Die Richtigkeit des Satzes ist eine unmittelbare Konsequenz des Hilfssatzes 1.2.

Die in Satz 3.10 definierten Zeichenfolgen ψ_i sind bis auf ψ_1 Präfixe von rechten Seiten von Produktionen. ψ_1 ist Präfix oder Infix einer rechten Seite einer Produktion, deswegen nennt man analog zu Def. 25 die Zeichenfolgen ψ_i eine Präfixdarstellung von $\# \gamma$.

Definition 3.15

Sei $G = (N, \Sigma, P, S)$ eine EPG, $\gamma \in \Gamma^*$ ein BU-Suffix von G , dann gilt: Jede Folge ψ_i , mit den in Satz 3.10 definierten Eigenschaften, heißt eine Präfixdarstellung von $\# \gamma$ (bzgl. G).

Nun können wir für BU-Suffixe die Mengen von gesuchten Variablen definieren.

Definition 3.16

Sei $G = (N, \Sigma, P, S)$ eine EPG, $\gamma \in \Gamma^*$ ein BU-Suffix von G , dann ist die Menge der für $\# \gamma$ gesuchten Variablen $\overline{\text{Ges}}(\# \gamma)$ definiert durch:

1. $\overline{\text{Ges}}(\# \gamma) = \{ S \}$ falls $\gamma = \epsilon$
2. $\overline{\text{Ges}}(\# \gamma) := \{ A \in N \mid \exists \gamma' \in \Gamma^* : \gamma' \gamma \text{ ist ein BU-Präfix von } G \text{ und } A \in \text{Ges}(\# \gamma') \}$ falls $\gamma \neq \epsilon$.

Satz 3.11

Sei $G = (N, \Sigma, P, S)$ eine EPG, $\gamma \in \Gamma^*$ ein BU-Suffix, ψ_i , $i = 0, \dots, k$ eine Präfixdarstellung von $\# \gamma$, $a \in \Sigma$, dann gilt:

1. Im Fall von $\text{End}(\psi_k) \hat{=} a$ gilt:

$$\gamma a \text{ ist BU-Suffix} \quad \Leftrightarrow \quad \exists A \in \overline{\text{Ges}}(\psi_0 \psi_1 \dots \psi_k) \text{ mit :} \\ \exists A \rightarrow \psi_k a \quad \lambda_k \in P, \lambda_k \in \Gamma^*$$

2. Im Fall von $\text{End}(\psi_k) \triangleleft a$ gilt:

γa ist BU-Suffix $\} \exists A \in \overline{\text{Ges}}(\psi_0 \dots \psi_k)$ mit:
 $\exists A \rightarrow \psi_k \subset \lambda_k \in P, C \in N, \lambda_k \in P^* \wedge$
 $C \xrightarrow{R}^* a \lambda, \lambda \in \Gamma^*$

3. Im Fall von $\text{End}(\psi_k) \triangleright a$ gilt:

- 3.1 Sei $k > 1$ und $B \in N$ die eindeutig bestimmte Variable mit $B \rightarrow \psi_1 \in P$, dann ist $\psi_1 \dots \psi_{k-1} B$ genau dann ein BU-Suffix, wenn $B \in \overline{\text{Ges}}(\psi_0 \dots \psi_k)$ ist.
- 3.2 Sei $k = 1$ und $B \in N$ eine Variable mit $B \rightarrow \lambda \psi_1 \in P, \lambda \in \Gamma^*$, dann ist B genau dann ein BU-Suffix, wenn $B \in \text{Ges}(\psi_0 \psi_1)$ ist.

Beweis:

Der Beweis ergibt sich durch triviale Anwendung der Definition von $\overline{\text{Ges}}(\psi_0 \dots \psi_k)$.

Beweis von (1)

"}"

Gelte $\text{End}(\psi_k) \triangleleft a$ und γa ist BU-Suffix von G }

$\exists \gamma' \in \Gamma^*$ mit $\gamma' \gamma a$ ist BU-Präfix von G }

$\exists A \in \text{Ges}(\# \gamma' \gamma a)$ mit $A \rightarrow \psi_k a \lambda_k \in P, \lambda_k \in \Gamma^*$ }

$A \in \overline{\text{Ges}}(\psi_0 \dots \psi_k)$.

q.e.d. ("}"")

"{"

Gelte $\text{End}(\psi_k) \triangleright a$ und $A \in \overline{\text{Ges}}(\psi_0 \dots \psi_k)$ mit $A \rightarrow \psi_k a \lambda_k \in P, \lambda \in \Gamma^*$

$\exists \gamma' \in \Gamma^*$ mit $A \in \text{Ges}(\# \gamma' \gamma)$ }

(Satz 2.6)

$\gamma'\gamma_a$ ist BU-Präfix von G } γ_a BU-Suffix von G

q.e.d. (" $\angle$ ")

q.e.d. (1)

Die Beweise von (2) und (3) sind völlig analog zu führen.

q.e.d. (Satz 3.11)

Für die Berechnung der Mengen $\overline{\text{Ges}}(\# \gamma)$ gilt der zu Satz 2.7 analoge Satz.

Satz 3.12

Sei $G = (N, \Sigma, P, S)$ eine EPG. $\gamma \in \Gamma^*$ ein BU-Suffix von G, ψ_i , $i = 0, \dots, k$ eine Präfixdarstellung von $\# \gamma$, $a \in \Sigma$, dann gilt:

1. Für $\gamma \in \Gamma$ gilt: $\overline{\text{Ges}}(\# \gamma) = \{ A \in N \mid \exists A \rightarrow \lambda_1 \gamma \lambda_2 \in P, \lambda_1, \lambda_2 \in \Gamma^* \}$
Andernfalls gelten:
2. Gilt $\text{End}(\gamma) \doteq a$ und ist γ_a ein BU-Suffix von G, dann folgt:

$$\overline{\text{Ges}}(\# \gamma_a) = \{ A \in N \mid A \in \overline{\text{Ges}}(\psi_0 \dots \psi_k) \wedge \exists A \rightarrow \psi_k a \lambda_k \in P, \lambda_k \in \Gamma^* \}$$

3. Gilt $\text{End}(\gamma) \neq a$ und ist γ_a ein BU-Suffix von G, dann folgt:

$$\overline{\text{Ges}}(\# \gamma_a) = \{ A \in N \mid \exists B \in \overline{\text{Ges}}(\psi_0 \dots \psi_k) \text{ mit: } \exists B \rightarrow \psi_k \bar{c} \lambda_k \in P, \lambda_k \in \Gamma^*,$$

$$\bar{c} \in N \wedge$$

$$C \xrightarrow{R}^* A \lambda, \lambda \in \Gamma^* \wedge$$

$$A \rightarrow a \lambda \in P, \lambda \in \Gamma^* \}$$

4. Gilt $\text{End}(\alpha) \triangleright a$ und $B \rightarrow \psi_k \in P$, wo $\psi_0 \dots \psi_{k-1} B$ ein BU-Suffix ist, dann folgt:

4.1 Falls $\text{End}(\psi_k) \doteq B$

$$\overline{\text{Ges}}(\psi_0 \dots \psi_{k-1} B) = \{ A \in N \mid \exists C \in \overline{\text{Ges}}(\psi_0 \dots \psi_{k-1}) \wedge C \rightarrow \psi_{k-1} B \lambda_{k-1} \in P, \\ \lambda_{k-1} \in \Gamma^* \}$$

4.2 Falls $\text{End}(\psi_k) \leq B$

$$\overline{\text{Ges}}(\psi_0 \dots \psi_{k-1} B) = \{A \in N \mid \exists C \in \overline{\text{Ges}}(\psi_0 \dots \psi_{k-1}) \wedge$$

$$C \rightarrow \psi_{k-1} \bar{C} \lambda_{k-1} \in P, \lambda_{k-1} \in \Gamma^* \wedge$$

$$\bar{C} \xRightarrow{R}^* A\lambda, \lambda \in \Gamma^* \wedge$$

$$A \rightarrow B\lambda \text{ } P, \lambda \in \Gamma^* \}$$

Der Beweis von Satz 3.12 ist eine triviale Erweiterung des Beweises von Satz 2.7.

Aus dem Satz 3.12 folgt, daß die Mengen $\overline{\text{Ges}}(\psi_0, \dots, \psi_k)$ schrittweise, parallel zur Syntaxanalyse berechnet werden können. Bezüglich des Implementierungs- und Laufzeitaufwandes gelten dieselben Überlegungen, wie sie im Anschluß an Satz 2.7 gemacht wurden.

Beispiel 3.17: Berechnung der gesuchten Variablen $\overline{\text{Ges}}(\psi_0 \dots \psi_k)$

Sei die Ll-Zeichenfolge

; if id :

gegeben. Die Teilzeichenfolge *; if id* wird von $\text{NSA}_{\overline{F3}}$ auf das BU-Suffix *L2 if id* reduziert.

Für die Mengen von gesuchten Variablen ergibt sich durch Anwendung von Satz 3.12:

$$\begin{aligned} \overline{\text{Ges}}(\#) &= \{ S \} \\ \overline{\text{Ges}}(\#;) &= \{ Ph, V, L2 \} \\ \overline{\text{Ges}}(\#L2) &= \{ L \} \\ \overline{\text{Ges}}(\#L2 \text{ if}) &= \{ I \} \\ \overline{\text{Ges}}(\#L2 \text{ if id}) &= \{ F \} \end{aligned}$$

Δ

Jede aus einem BU-Suffix mit Hilfe von Rechtsableitungen gewonnene Zeichenfolge $u \in \Sigma^*$ ist ein Infix eines Wortes der zugrunde liegenden Sprache. Deshalb ist Satz 3.11 die Grundlage für die Konstruktion einer Analysefunktion $\overline{F4}$ die einen Nachsilbenanalysator $\text{NSA}_{\overline{F4}}$ induziert, welcher die LP-Eigenschaft hat.

Satz 3.13

Sei $G = (N, E, P, S)$ eine EPG, $L = L(G)$, Konf_G die Menge der Konfigurationen

über G, dann gilt:

Die Abbildung $\overline{F4} : \text{Konf} \rightarrow 2^{\text{Konf}_G}$, definiert durch:

$$1. \overline{F4}(\#u\#) := (\#a, u'\#) \quad \text{falls } u = au', a \in \Sigma, u' \in \Sigma^*$$

$$2. \overline{F4}(\#\alpha, u\#) := \left\{ \begin{array}{ll} (\#\alpha a, u') & \text{falls} \quad (1) \\ & 1. u\# = au', a \in \Sigma, u' \in \Sigma^* \wedge \text{End}(\alpha) \neq a \\ & 2. \exists A \in \overline{\text{Ges}}(\psi_0 \dots \psi_k) : A \rightarrow \psi_k a \lambda_k \in P, \\ & \quad \lambda_k \in \Gamma^* \\ (\#\alpha a, u') & \text{falls} \quad (2) \\ & 1. u\# = au', a \in \Sigma, u' \in \Sigma^* \wedge \text{End}(\alpha) \neq a \\ & 2. \exists A \in \overline{\text{Ges}}(\psi_0 \dots \psi_k) : A \rightarrow \psi_k C \lambda_k \in P \wedge \\ & \quad C \xrightarrow{R^*} a \lambda, \lambda, \lambda_k \in \Gamma^* \\ (\#\psi_1 \dots \psi_{k-1} B, u\#) & \text{falls} \quad (3) \\ & 1. \text{End}(\#\psi_1 \dots \psi_k) \triangleright \text{Anf}(u\#) \\ & 2. k > 1 \\ & 3. \exists B \rightarrow \psi_k \in P : B \in \overline{\text{Ges}}(\psi_0 \dots \psi_k) \\ (\#B, u\#) & \text{falls} \quad (4) \\ & 1. \text{End}(\#\psi_1 \dots \psi_k) \triangleright \text{Anf}(u\#) \\ & 2. k = 1 \\ & 3. \exists B \rightarrow \lambda_k \psi_1 \in P : B \in \overline{\text{Ges}}(\psi_0 \dots \psi_1), \\ & \quad \lambda_k \in \Gamma^* \\ \text{true} & \text{falls } k = 1 \wedge \psi_1 = S \wedge u = \varepsilon \quad (5) \\ |u\#| - 1 & \text{sonst} \quad (6) \end{array} \right.$$

falls $\alpha \in \Gamma^*$ ein BU-Suffix von G,
 $\psi_i, i=0, \dots, k$ eine Präfixdarstellung von $\# \alpha$
 und $u \in \Sigma^*$ ist

eine Nachsilbenanalysefunktion für L mit der Eigenschaft:

Sei $u \in \Sigma^+$ mit $(\#u\#) \overline{F4}^+(\#\alpha, u'\#)$, dann ist α ein BU-Suffix von L.

Beweis:

Da $\overline{F_4}$ sich von $\overline{F_1}$ nur hinsichtlich des Zeitpunkts der Erkennung von Syntaxfehlern unterscheidet, ist $\overline{F_4}$ eine Nachsilbenanalysefunktion. Der Rest der Behauptung folgt direkt aus Satz 3.11.

q.e.d.

Bemerkung 3.5

In den Fällen (1) - (4) des obigen Satzes wird geprüft, ob der zu erstellende Kellerinhalt ein BU-Suffix ist. Deshalb bezeichnen wir im Fehlerfall die entsprechenden Fehlertypen mit BUS1 - BUS4.

Der folgende Satz ist eine triviale Konsequenz der Sätze 3.13 und 3.3 .

Satz 3.14

Sei $G = (N, \Sigma, P, S)$ eine EPG, dann hat der von der Nachsilbenanalysefunktion $\overline{F_4}$ induzierte Analysator $NSA_{\overline{F_4}}$ die LP-Eigenschaft für L, und der von $A_{\overline{F_4}}$ und $NSA_{\overline{F_4}}$ induzierte Analysator $\mathcal{U}_{A_{\overline{F_4}}, NSA_{\overline{F_4}}}$ bildet jedes Element $x \in \Sigma^* \setminus L$ auf eine minimale Nachsilbenanalyse bzgl. L ab.

Beispiel 3.18: Analyse mit Hilfe von $\mathcal{U}_{A_{\overline{F_4}}, NSA_{\overline{F_4}}}$

Die Analyse der Zeichenfolge u aus Beispiel 3.15 wird durch die folgende Tabelle wiedergegeben.

Fehler	Kellerinhalt bei Fehlererkennung	Relation	akt. Zeichen	Fehlertyp	FA
F1	$\# \leftarrow Ph \leftarrow begin \leftarrow id \stackrel{!}{=} [\leftarrow (\stackrel{!}{=} E$	$\stackrel{!}{=}$	]	RS1	+
F2	$\# L2 \leftarrow id$	$\triangleright$	;	BUS3	<i>while</i>
F3	$\# while \stackrel{!}{=} E$	$\stackrel{!}{=}$	<i>then</i>	RS2	<i>id</i>
F4	$\# id \stackrel{!}{=} G \leftarrow E$	$\stackrel{!}{=}$	)	RS1	;
F5	$\# L2 \leftarrow if \leftarrow id$	$\stackrel{!}{=}$	:	BUS1	<i>id</i>
F6	$\# I \leftarrow id$	$\triangleright$	=	BUS3	<i>id</i>

Die durch $\mathcal{U}_{A_{F4}, NSA_{F4}}$ erzeugte Zerlegung x_i $i=1, \dots, 7$ von u ist also durch

$x_1 = \text{program } id ; id := id [(id + id]$
 $x_2 = + id * id ; id ;$
 $x_3 = while id then$
 $x_4 = id := id)$
 $x_5 = ; if id :$
 $x_6 = id + id then id =$
 $x_7 = id end end .$

gegeben. Alle Fehler werden frühestmöglich erkannt, jeweils das letzte Zeichen jeder Zeichenfolge ist das, die Fehlererkennung auslösende Zeichen.

Δ

Durch Satz 3.14 ist das Problem der Konstruktion eines fehlerbehandelnden Analysators, welcher jede fehlerhafte Zeichenfolge auf eine minimale Nachsilbenanalyse abbildet, für einfache Präzedenzsprachen theoretisch gelöst. Gleichzeitig wurde eine Möglichkeit der Implementierung des Analysators in Grundzügen beschrieben. Die beiden Hauptprobleme dabei waren

die Berechnung der Mengen von gesuchten Variablen und die Erweiterung des Präzedenzanalysealgorithmus um die Möglichkeit, in gewissen Fällen den Eingabezeiger zurückzusetzen bzw. parallel zu analysieren. Von dem ersten Problem konnte gezeigt werden, daß es ohne größeren Implementierungs- und Laufzeitaufwand gelöst werden kann. Die Lösung des zweiten Problems bereitet ebenfalls keine implementierungstechnischen Schwierigkeiten. Die Abschätzung des Zeit- bzw. Platzbedarfs des Analysators $NSA_{\overline{F4}}$ in Abhängigkeit von der Länge der Eingabezeichenfolge ist dagegen nicht trivial. Er hängt genau dann nicht linear von der Länge der Eingabezeichenfolge ab, wenn es eine einfache Präzedenzsprache L über Σ gibt, welche eine Folge f , gegeben durch $x_n \in \Sigma^*$, $n \in \mathbb{N}$, enthält mit der Eigenschaft:

$$\text{Zu } k \in \mathbb{N} \exists x_{i_k} \in f : \left| \{ (\# \alpha, u \#) \mid (\#, x_{i_k} \#) \overline{F4} (\# \alpha, u \#) \} \right| > k.$$

Realisiert man den Analysator $NSA_{\overline{F4}}$ mit Hilfe von simultan gehaltenen Kellern, so gibt es keine obere Grenze für die Anzahl der dazu benötigten Keller. Daraus folgt, daß der Zeit- bzw. Platzbedarf mindestens quadratisch von der Länge der Eingabezeichenfolge abhängt.

Beispiel 3.19: Zeit- bzw. Platzbedarf von $NSA_{\overline{F4}}$

Die Sprache L sei definiert durch die Produktionen:

$$\begin{aligned} S &\rightarrow F \\ F &\rightarrow a \\ F &\rightarrow v B a \\ F &\rightarrow F G \\ G &\rightarrow F y \\ B &\rightarrow x a \\ B &\rightarrow u B a \end{aligned}$$

L ist eine einfache Präzedenzsprache. Die Präzedenzrelationen sind in der folgenden Matrix wiedergegeben:

	S	F	G	B	a	u	v	x	y
S									
F									
G									
B									
a									
u									
v									
x									
y									

Sei die Folge x_n definiert durch:

$$x_n := a^n y^{n-1}, \quad n \in \mathbb{N}$$

, dann gilt für $\alpha_0 := B$ und $\alpha_i := F^i$, $i = 1, \dots, n$:

$$(\#, x_n \#) \overline{F4} (\# \alpha_i y^{n-1} \#) \text{ für } i = 1, \dots, n$$

Die obige Behauptung kann mit Hilfe von vollständiger Induktion über n bewiesen werden. Dies wird unmittelbar klar, wenn wir untersuchen, wie zum Beispiel die Zeichenfolge x_4 mit Hilfe von $\overline{F4}$ analysiert wird.

$$\begin{aligned} \overline{F4}(\#, aaayyy \#) &= (\#a, aaayyy \#) \\ \overline{F4}(\#a, aaayyy \#) &= \{(\#F, aaayyy \#), (\#B, aaayyy \#)\} \end{aligned}$$

, da $a \triangleright a$ und $F \rightarrow a \in P$ und $B \rightarrow uBa \in P$ gilt.

$$\overline{F4}(\#F, aaayyy \#) = (\#Fa, aaayyy \#), \text{ da } F \triangleleft a \text{ gilt.}$$

$$\overline{F4}(\#B, aaayyy \#) = (\#Ba, aaayyy \#), \text{ da } B \perp a \text{ gilt.}$$

$$\overline{F4}(\#Fa, aaayyy \#) = (\#FF, aaayyy \#) \text{ und}$$

$$\overline{F4}(\#FF, aaayyy \#) = (\#FFa, aaayyy \#), \text{ da } F \rightarrow a \in P \text{ gilt.}$$

$$\overline{F4}(\#Ba, aaayyy \#) = \{(\#B, aaayyy \#), (\#F, aaayyy \#)\}$$

da $a \triangleright a$ und $B \rightarrow uBa \in P$ und $F \rightarrow vBa$ gilt.

usw.

Die Mehrdeutigkeiten, die bei der Analyse der Zeichenfolge x_n auftreten werden dadurch verursacht, daß die Sprache L eine Klammerstruktur enthält,

bei der die öffnende Klammer selbst eine Klammerstruktur ist, und das Zeichen für die schließende Klammer dieser Klammerstruktur, gleichzeitig öffnende Klammer in der ersten Klammerstruktur ist.

Es gilt:

$$F \Rightarrow^+ F^k y^{k-1}$$

$$F \rightarrow a \in P$$

$$F \Rightarrow^+ vu^1 xa^1 a^1$$

Das heißt, die Variable F und das Zeichen y bilden ein Klammerpaar.

Wird F nach a abgeleitet, so ist a öffnende Klammer.

Wird F nach $vu^1 xa^1 a^1$ abgeleitet, so ist a schließende Klammer.

Analysiert die Nachsilbenanalysefunktion $\overline{F4}$ eine Zeichenfolge der Bauart x_n , so kann sie die Entscheidung ob ein Zeichen a öffnende oder schließende Klammer ist erst fällen, wenn sie die Anzahl der Zeichen y kennt.

Δ

Das obige Beispiel zeigt, daß es ein interessantes und schwieriges Problem ist, hinreichend große Sprachklassen zu definieren, bei denen Zeichenfolgen x_n mit den genannten Eigenschaften nicht auftreten können. Da verbreitete Programmiersprachen wie ALGOL60 und PASCAL keine Sprachstrukturen enthalten, wie sie in Beispiel 3.19 vorkommen, ist die Lösung dieses Problems nur von theoretischer Bedeutung und wird in dieser Arbeit nicht weiter verfolgt.

4. ÜBERTRAGUNG DER BISHER GEWONNENEN ERGEBNISSE AUF ANALYSATOREN, WELCHE PRÄZEDENZFUNKTIONEN BENÜTZEN

Der Platzbedarf zur Speicherung der Präzedenzrelationen ist proportional n^2 , wenn n die Anzahl der Elemente des Alphabets der zugrunde liegenden Grammatik ist. Definiert man die Syntax einer höheren Programmiersprache, wie zum Beispiel PASCAL mit Hilfe einer einfachen Präzedenzgrammatik, so ergeben sich für n Werte in der Größenordnung zwischen zwei- und vierhundert. (bei PASCAL ungefähr 240, siehe [Rho,73]). Das bedeutet, daß in einem solchen Fall die Speicherung der Relationen ein erhebliches Problem bei der Realisierung eines Analysators ist.

Die theoretisch beste Möglichkeit, diesen Speicherplatz drastisch zu verkleinern, ist die Verwendung von Präzedenzfunktionen, falls solche existieren. In diesem Fall ist der Platzbedarf proportional $2 \cdot n$. Die bisher bekannten Analysatoren, welche auf der Verwendung von Präzedenzfunktionen beruhen sind jedoch für Anwendungen in der Praxis nicht geeignet, da sie syntaktische Fehler i.allgemeinen erst sehr spät erkennen, und dadurch eine sinnvolle Behandlung der Fehler i.allg. nicht möglich ist. Aus diesem Grund sind keine Fehlerbehandlungsmethoden für solche Analysatoren bekannt. (Siehe [Aho.Ull,73], Seite 563).

In diesem Abschnitt wird gezeigt werden, daß die im Abschnitt 2 entwickelte Methode der Fehlererkennung auf Analysatoren übertragen werden kann, welche Präzedenzfunktionen benutzen. Eine unmittelbare Konsequenz daraus ist, daß auch die im Abschnitt 3 entwickelten Fehlerbehandlungsmethoden auf diese Analysatoren übertragen werden können.

4.1 Analyse mit Hilfe von Präzedenzfunktionen

In diesem Abschnitt definieren wir den Begriff Präzedenzfunktionen. Gleichzeitig zeigen wir, wie die in dieser Arbeit definierten Analysatoren erweitert werden können zu Analysatoren, welche Präzedenzfunktionen benützen.

Definition 4.1

Sei $G = (N, \Sigma, P, S)$ eine EPG, f, g seien Abbildungen von $\Gamma \cup \{\#\}$ nach $\mathbb{N}$, dann gilt:

f, g heißen Präzedenzfunktionen für G : $\forall$ Für $X, Y \in \Gamma$ gilt:

1. $X \doteq Y \quad \} \quad f(X) = g(Y)$
2. $X \triangleleft Y \quad \} \quad f(X) < g(Y)$
3. $X \triangleright Y \quad \} \quad f(X) > g(Y)$
4. $f(\#) < g(Y) \wedge f(X) > g(\#)$

Bemerkung 4.1

Die Forderung (4) wurde hinzugenommen, um die Formulierung von Analysefunktionen zu vereinfachen. Sie stellt keine Beschränkung der Allgemeinheit dar.

Als Beispiel zu der Definition 4.1 werden in Bild 4.1 die Präzedenzfunktionen f_1 und g_1 für die Grammatik G_1 angegeben. Sie wurden mit Hilfe des Algorithmus von Floyd [Flo,63] berechnet.

Bild 4.1: Die Präzedenzfunktionen $f1$, $g1$ für $G1$

X	$f1(X)$	$g1(X)$
<i>S</i>	1	1
<i>Pi</i>	2	2
<i>Ph</i>	1	2
<i>B</i>	2	1
<i>V</i>	2	2
<i>Vh</i>	2	2
<i>IL1</i>	2	1
<i>IL</i>	4	2
<i>Ch</i>	5	2
<i>C</i>	2	4
<i>L1</i>	4	1
<i>L2</i>	1	2
<i>L</i>	2	2
<i>S</i>	2	2
<i>Os</i>	3	3
<i>Cs</i>	3	3
<i>Ic</i>	4	4
<i>I</i>	3	4
<i>W</i>	3	4
<i>A</i>	1	4
<i>G</i>	4	5
<i>E</i>	1	1
<i>Th</i>	4	2
<i>T1</i>	2	2
<i>T</i>	5	3
<i>F</i>	6	3
<i>program</i>	7	2
<i>var</i>	4	2
<i>id</i>	1	4
<i>begin</i>	7	4
<i>end</i>	1	1
<i>do</i>	5	1
<i>else</i>	3	3
<i>if</i>	3	4
<i>then</i>	1	4
<i>while</i>	5	4
<i>(</i>	1	4
<i>)</i>	1	4
<i>[</i>	7	7
<i>]</i>	1	4
<i>+</i>	7	6
<i>*</i>	3	6
<i>=</i>	3	6
<i>:</i>	5	7
<i>.</i>	2	4
<i>;</i>	5	2
<i>,</i>	2	7
<i>#</i>	1	1

Wollen wir die in dieser Arbeit definierten Analysatoren für einfache Präzedenzsprachen zu Analysatoren abändern, welche Präzedenzfunktionen benutzen, so können wir dies dadurch tun, daß wir in den entsprechenden Analysefunktionen die Präzedenzrelationen $\hat{=}$, $\hat{<}$, $\hat{>}$ zwischen den Zeichen des Alphabets ersetzen durch die arithmetischen Relationen $=$, $<$, $>$ zwischen den entsprechenden Werten der Präzedenzfunktionen. Im folgenden Satz wird auf diese Weise aus der Analysefunktion $F1$ die Analysefunktion $\tilde{F1}$ gewonnen.

Satz 4.1

Sei $G = (N, L, P, S)$ eine EPG, Konf_G die Menge der Konfigurationen über G und f, g Präzedenzfunktionen für G , dann gilt:

Die Abbildung $\tilde{F1} : \text{Konf}_G \rightarrow \text{Konf}_G$ definiert durch

$$\tilde{F1}(\#X_1X_2\dots X_i, u\#) := \left\{ \begin{array}{ll} (\#X_1\dots X_i a, \tilde{u}\#) \text{ falls} & (1) \\ \quad 1. u = a\tilde{u}, \tilde{u} \in \Sigma^*, a \in \Sigma \\ \quad 2. f(X_i) = g(a) \vee f(X_i) < g(a) & \\ \\ (\#X_1\dots X_p A, u\#) \text{ falls} & (2) \\ \quad 1. f(X_i) > g(\text{Anf}(u\#)) \\ \quad 2. \exists p \in \mathbb{N}^0 : p = \max_{m=0, \dots, i-1} (f(X_m) < g(X_{m+1})) \\ \quad 3. \exists A \rightarrow X_{p+1}\dots X_i \in P & \\ \\ \text{true} & \text{falls } i=1 \wedge X_i = S \wedge u = \epsilon \quad (3) \\ \\ |u\#| - 1 & \text{falls} \quad (4) \\ \quad 1. f(X_i) > g(\text{Anf}(u\#)) \\ \quad 2. \exists p \in \mathbb{N}^0 : p = \max_{m=0, \dots, i-1} (f(X_m) < g(X_{m+1})) \\ \quad 3. \nexists A \rightarrow X_{p+1}\dots X_i \in P \end{array} \right.$$

, wo $X_j \in \Gamma$ für $j = 1, \dots, i$, $X_0 := \#$, $u \in \Sigma^*$ gilt,

ist eine Analysefunktion für $L = L(G)$.

Beweis:

Der Beweis des Satzes folgt unmittelbar aus Satz 1.2 und der Definition der Präzedenzfunktionen. Δ

Vergleicht man die Analysefunktion $\widetilde{F1}$ mit der Analysefunktion $F1$ (siehe Satz 1.2), so zeigt sich der bereits erwähnte Nachteil der Analyse mit Hilfe von Präzedenzfunktionen. $\widetilde{F1}$ erkennt alle Fehler in der Form von Reduktionsfehlern. Zeichenpaarfehler bzw. Typ1-Fehler werden nicht als solche erkannt. Die Ursache dafür ist, daß durch die Verwendung von Präzedenzfunktionen nicht mehr festgestellt werden kann, ob zwischen zwei Zeichen des Alphabets eine Präzedenzrelation besteht oder nicht.

Beispiel 4.1: Fehlererkennung bei Analyse mit Hilfe von Präzedenzfunktionen ($A_{\widetilde{F1}}$, von $\widetilde{F1}$ induzierter Analysator)

Sei die bezüglich $L1$ fehlerhafte Zeichenfolge u durch

```

u :=  program id ;
      var ( id + id ) ;
          ↑   ↑   ↑
      begin
          id := id
      end.

```

gegeben. Sie enthält drei Fehler (markiert durch $\uparrow$). Vor dem Lesen von (hat $\widetilde{F1}$ den Kellerinhalt

$\# Pi \quad var$

erstellt. Wegen $fl(var) = gl(() = 4$ wird (gekellert und die Analyse fortgesetzt. Die Existenz eines Fehlers wird erst beim Versuch der Reduktion der Zeichenfolge

$var (E)$

erkannt. Δ

Es ist unmittelbar klar, daß auf analoge Weise aus den Analysefunktionen

F2 und F3 Analysefunktionen $\widetilde{F2}$ und $\widetilde{F3}$ gewonnen werden können. $\widetilde{F2}$ ist identisch mit $\widetilde{F1}$, da die Prüfung auf Kellerungsfehler bei Verwendung von Präzedenzfunktionen sinnlos ist. Durch die Einführung der Prüfung auf RS-Fehler ist die Fehlererkennung von $\widetilde{F3}$ gegenüber der von $\widetilde{F1}$ zwar erheblich, doch nicht entscheidend verbessert. Deshalb ist es ein interessantes Problem, ob aus Analysefunktion F4 eine Analysefunktion $\widetilde{F4}$ gewonnen werden kann, so daß der von $\widetilde{F4}$ induzierte Analysator die LP-Eigenschaft hat. Damit beschäftigt sich der nächste Abschnitt.

4.2 Übertragung der frühestmöglichen Fehlererkennung

Die Erweiterung von F4 zu $\widetilde{F4}$ ist eine triviale Konsequenz der im Abschnitt 2 bewiesenen Eigenschaften von Mengen von gesuchten Variablen.

Satz 4.2

Sei $G = (N, \Sigma, P, S)$ eine EPG, $L = L(G)$, Konf_G die Menge der Konfigurationen über G_1 , f, g Präzedenzfunktionen für G , dann gilt:

Die Abbildung $\widetilde{F4} : \text{Konf}_G \rightarrow \text{Konf}_G$, definiert durch:

$$\widetilde{F4}(\# \alpha, u \#) := \begin{cases} (\# \alpha, u') & \text{falls} & (1) \\ & 1. f(\text{End}(\# \alpha)) = g(a) \\ & 2. \exists A \in \text{Ges}(\psi_0 \dots \psi_k) : \\ & \quad A \rightarrow \psi_k a \lambda_k \in P, \lambda_k \in \Gamma^* \\ (\# \alpha, u') & \text{falls} & (2) \\ & 1. f(\text{End}(\# \alpha)) < g(a) \\ & 2. \exists B \in \text{Ges}(\psi_0 \dots \psi_k) : B \rightarrow \psi_k C \lambda_k \in P \\ & \quad \wedge C \xrightarrow{R^*} a \lambda, \lambda, \lambda_k \in \Gamma^* \\ (\# \psi_1 \dots \psi_{k-1} B, u \#) & \text{falls} & (3) \\ & 1. f(\text{End}(\# \psi_1 \dots \psi_k)) > g(a) \\ & 2. \exists B \in \text{Ges}(\psi_0 \dots \psi_k) : B \rightarrow \psi_k \in P \\ \text{true} & \text{falls } k = 1 \wedge \psi_1 = S \wedge u = \varepsilon & (4) \\ |u \#| - 1 & \text{sonst} & (5) \end{cases}$$

, wo α ein BU-Präfix, $\psi_0 \dots \psi_k$ eine Präfixdarstellung von $\# \alpha$ ist und $u \# = au'$, $u' \in \Sigma^* \#$, $a \in \Sigma$ gilt,

ist eine Analysefunktion für mit der Eigenschaft:

Sei $u \in \Sigma^*$ mit $(\#, u \#) \in \widetilde{F4}^+$ ($\# \alpha, u' \#$), dann ist α ein BU-Präfix von G.

Beweis:

Die Behauptung des Satzes folgt auf triviale Weise aus Satz 2.6, denn:

In den Fällen (1) - (3) der Definition von $\widetilde{F4}$ folgt aus der Gültigkeit der zweiten Bedingung (durch Anwendung von Satz 2.6), daß αa bzw.

$\psi_1 \dots \psi_{k-1}^B$ ein BU-Präfix ist. Daraus folgt die Behauptung.

q.e.d.

Aus Satz 4.2 folgt unmittelbar, daß der von $\widetilde{F4}$ induzierte Analysator die LP-Eigenschaft besitzt. Das heißt, wir haben einen Analysator für einfache Präzedenzsprachen gewonnen, welcher Präzedenzfunktionen benutzt und dennoch die LP-Eigenschaft hat.

Im nächsten Abschnitt wird gezeigt werden, daß die in Abschnitt drei entwickelten Fehlerbehandlungsmethoden auf analoge Weise übertragen werden können.

4.3 Übertragung der Fehlerbehandlung

Die in Abschnitt 3.3.3.1 durchgeführten Überlegungen gelten auch dann, wenn die zugrunde liegenden Analysatoren Präzedenzfunktionen statt den Präzedenzrelationen verwenden. Ersetzt man deshalb in den Abbildungen $F1$ bis $F4$ die Präzedenzrelationen durch die entsprechenden arithmetischen Relationen, so gewinnt man die Nachsilbenanalysefunktionen $\widetilde{F1}$ bis $\widetilde{F4}$, welche mit Hilfe von Präzedenzfunktionen analysieren. Der von $A_{\widetilde{F4}}$ und $NSA_{\widetilde{F4}}$ induzierte fehlerbehandelnde $\mathcal{U}_{A_{\widetilde{F4}}, NSA_{\widetilde{F4}}}$ bildet jede Eingabezeichenfolge auf eine minimale Nachsilbenanalyse ab.

5. ZUSAMMENFASSUNG

Ziel dieser Arbeit war es, für einfache Präzedenzsprachen einen fehlerbehandelnden Analysealgorithmus zu entwickeln, welcher das Prinzip des zeichengesteuerten Wiederaufsetzens in dem Fall verwirklicht, in dem alle terminalen Zeichen als Wiederaufsetzzeichen verwendet werden.

Es wurden die folgenden Ergebnisse erzielt:

1. Es wurde, ausgehend von Aho und Ullman, eine Methode der Beschreibung von Analysealgorithmen für einfache Präzedenzsprachen entwickelt, mit deren Hilfe die Eigenschaften der in dieser Arbeit vorgeschlagenen Analysealgorithmen formal bewiesen werden konnten. Diese Beschreibungsmethode konnte so gewählt werden, daß aus der formalen Beschreibung auf die Realisierung der Algorithmen in Form von Programmen geschlossen werden kann. Ferner wurde darauf hingewiesen, daß diese Beschreibungsmethode, nach entsprechenden Modifikationen, zur formalen Darstellung von Bottom-Up Analysatoren im allgemeinen verwendet werden kann.
2. Es wurde ein Analysealgorithmus für einfache Präzedenzsprachen entworfen, der syntaktische Fehler frühestmöglich erkennt. Die Existenz eines solchen Analysealgorithmus war für die in dieser Arbeit vorgeschlagene Realisierung des gesuchten Wiederaufsetzverfahrens notwendig.
3. Das gesuchte zeichengesteuerte Wiederaufsetzverfahren konnte konstruiert werden. Seine wesentliche Eigenschaft ist, daß es jede Eingabezeichenfolge in eine minimale Folge von Infizen der Sprache zerlegt. Dies ist ein neuer Ansatz zur Lösung des Fehlerbehandlungsproblems, der sich von den bisher bekannten Lösungssätzen dadurch unterscheidet, daß weder implizit noch explizit ein Fehlerbegriff benutzt wird. Es konnte ferner gezeigt werden, daß das Verfahren die in der Einlei-

tung geforderten Eigenschaften E1 bis E3 in, für Anwendungen in der Praxis hinreichender Weise erfüllt, und insbesondere keine Folgefehler meldet.

4. Alle in dieser Arbeit erzielten Ergebnisse für Analysatoren für einfache Präzedenzsprachen, welche mit Präzedenzrelationen arbeiten, konnten übertragen werden auf solche Analysatoren, welche mit Präzedenzfunktionen arbeiten. Dadurch ist das Problem der Erweiterung solcher Analysatoren um einen Algorithmus zur Behandlung von Syntaxfehlern erstmalig gelöst.
5. Es wurden die theoretischen Grundlagen gegeben, mit Hilfe derer effiziente Implementierungen für die in dieser Arbeit vorgeschlagenen Analysealgorithmen gewonnen werden können. Die Implementierungen selbst wurden nur prinzipiell beschrieben. Deshalb konnte die Beurteilung der Frage inwieweit Implementierungen die in E4 geforderten Eigenschaften erfüllen, nur heuristisch durchgeführt werden. Dabei ergab sich, daß die Forderung E4 befriedigend erfüllt werden dürfte.

6. LITERATURVERZEICHNIS

Das folgende Verzeichnis gibt einen Überblick über die wichtigste Literatur zum Problem der Behandlung von Syntaxfehlern. Arbeiten, auf die im Verlauf dieser Arbeit Bezug genommen wird, sind dadurch gekennzeichnet, daß ihre symbolische Bezeichnung am linken Rand notiert wird.

- | | | |
|--------------|----------------------------|---|
| | Alberga,C.N. | : String Similarity and Misspellings
CACM, Vol.10, No.5, May 1967 |
| [Amm,75] | Amman,U. | : Die Entwicklung eines PASCAL-Compilers
nach der Methode des Strukturierten
Programmierens
Diss. ETH 5456, 1975 |
| | Aho,A.V.,
Johnson,S.C. | : LR Parsing
Computing Surveys, Vol.6, No.2,
June 1974 |
| [Aho.Pet,72] | Aho,A.V.,
Peterson,T.G. | : A Minimum Distance Error-Correcting
Parser for Context-Free Languages
SIAM Journal of Computing,
Vol.1, No.4, Dec.1972 |
| [Aho.Ull,72] | Aho,A.V.,
Ullman,J.P. | : The Theory of Parsing, Translation and
Compiling, Volume 1: Parsing
Prentice-Hall 1972 |
| | Aho,A.V.,
Ullman,J.P. | : Error Detection in Precedence Parsers
Math. Systems Theory, Vol.7, No.2,1973 |
| | Aho,A.V.,
Ullman,J.P. | : Principles of Compiler Design
Addison - Wesley Publ.Comp., 1977 |

- Barnard,D.T.: : Automatic Generation of Syntax-
Repairing and Paragraphing Parsers
Techn.Rep. CSRG-52, April 1975, Univ.of
Toronto
- Ciesinger,J. : Generating Error Recovery in a
Compiler Generating System
4. Fachtagung der GI, 1976
Informatik-Fachberichte 1, Springer
- [Con.Max,63] Conway,R.W., : CORC - The Cornell Computing Language
Maxwell,W.L. CACM, Vol.6, No.6, June 1963
- Conway,R.W., : Design and Implementation of a Diagnostic
Wilcox,T.R. Compiler for PL/I
CACM, Vol.16, No.13, March 1973
- [Dam,64] Damerau,F. : A Technique for Computer Detection
and Correction of Spelling Errors
CACM, Vol.7, No.3, March 1964
- Dieterich,E.-W. : Parsing and Syntactic Error Recovery
for Context-Free Grammars by Means
of Coarse Structures
Fourth Coll. on Automata, Languages
and Programming, 1977
Springer Lecture Notes 52
- Druseikis,F.G., : Error Recovery For Simple LR(k) Parsers
Ripley,C.G. Proceedings of the ACM 76 Conference,
Houston 1976
- Eggers,B. : Zur Theorie und Praxis selbstkorrigie-
render regulärer Sprachen
Diss.Techn.Universität Hannover 1971

- Eggers, B. : Error Reporting, Error Treatment, and
Error Correction in Algol Translation
Part 2
GI 2. Jahrestagung, Proceedings 1972
- Elspas, B., : Software Reliability
Green, M.W., Computer 1, 1971
Levitt, K.N.
- Feyock, St., : Syntax-Directed Correction of Syntax
Lazarus, P. Errors
Software - Practice and Experience
Vol. 6, 1976
- [Flo, 63] Floyd, R.W. : Syntactic Analysis and Operator Precedence
J. ACM 10, 3, 1963
- [Fre, 64] Freemann, D. : Error Correction in CORC
Proceedings - Fall Joint Computer
Conference, 1964
- Ghezzi, C. : LL(1) Grammars Supporting an Efficient
Error Handling
Information Processing Letters,
Vol. 3, No. 6, July 1975
- Graham, L.S., : Practical Syntactic Error Recovery in
Rhodes, St. P. Compilers
Proceedings of Symposium on Principles
of Programming Languages, Boston,
Oct. 1973
- [Gri, 71] Gries, D. : Compiler Construction
John Wiley & Sons, New York, 1971

- | | | |
|----------|--------------------------------|---|
| [Gri,75] | Gries,D. | : Error Recovery and Correction -
An Introduction to the Literature
Advanced Course on Compiler
Construction, Munich, March 1975 |
| [Hau,78] | Hauger,F. | : Ein neues Verfahren zur Behandlung
von syntaktischen Fehlern bei einfa-
chen Präzedenzsprachen
Diplomarbeit (in Vorbereitung),
TU München, 1978 |
| | Hext,J.B. | : Recovery from Error
Computer and Automation, 16, 1967 |
| | Hopcroft, J.E.,
Ullman,J.D. | : Error Correction for Formal Languages
TR No.52, Nov. 1966, Princeton Univ. |
| | Horning,J.J. | : What the Compiler Should Tell The User
Compiler Construction - An Advanced
Course,
Lecture Notes in Computer Science 21,
Springer, 1975 |
| | Irons,E.T. | : An Error-Correcting Parse Algorithm
CACM, Vol.6, No.11, Nov. 1973 |
| [Jam,73] | James,L.R. | : A Syntax Directed Error Recovery
Method
TR CSRG-13, 1973, Univ. of Toronto |
| | James,E.R.,
Partridge,D.P. | : Adaptive Correction of Program Statements
CACM, Vol.6, Num.1, Jan.1973 |
| | Koster,C.H.A. | : Error Reporting, Error Treatment, and
Error Correction in Algol Translation -
Part.I
GI 2.Jahrestagung Proceedings 1972 |

- [LaF,71] LaFrance,J.E : Syntax-Directed Error Recovery for
Compilers
PhD Thesis 1971, Univ.of Illinois
- LaLonde,W.R. : An Efficient LALR Parser Generator
TR CSRG-2, 1971, Univ.of Toronto
- [Lei,70] Leinius,R.P. : Error Detection and Recovery for
Syntax Directed Compiler Systems
PhD Thesis, 1970, Univ.of Wisconsin
- [Lév,71] Lévy,J.P. : Automatic Correction of Syntax Errors
in Programming Languages
PhD Thesis, 1971, Cornell University
- Lévy,J.P. : Automatic Handling of Syntax-Errors
in SCARABEE, an Interactive System
Cahier de Recherche No.2, 1973
Centre D'Enseignement des Affaires
- [Lyo,74] Lyon,G. : Syntax-Directed Least Errors Analysis
for Context-Free Languages: A Practical
Approach
CACM, Vol.17, No.1, January 1974
- [Mau,69] Maurer,H. : Theoretische Grundlagen der Programmier-
sprachen
B.I. Hochschultaschenbuch 404/404a
- McGruther,T. : An Approach to Automatic Syntax Error
Detection, Recovery and Correction
for LR(k) Grammars
Masters Thesis, Naval Postgraduate
School, Monterey, Calif., 1972

- Morgan,H.L. : Spelling Correction in Systems Programs
CACM, Vol.13, No.2, Febr.1970
- Moulton,P.G., : DITRAN - A Compiler Emphasizing
Muller,M.E. Diagnostics
CACM, Vol.10, No.1, Jan.1967
- Parnas,D.L. : Response to Detected Errors in Well-
Structured Programs
Technical Report, Carnegie-Mellon Univ.
September, 1972
- Peterson,T.G. : Syntax Error Detection, Correction and
Recovery in Parsers
Thesis 1972, Stevens Inst.of Techn.,
N.J.
- Poonen,G. : Error Recovery for LR(k) Parsers
Proceedings of IFIP Congress 77
North Holland
- [Rho,73] Rhodes,St.P. : Practical Syntactic Error Recovery
for Programming Languages
PhD Thesis, 1973, Univ.of Calif.,
Berkeley
- Roehrich,J. : Syntax-error Recovery in LR-Parsers
4. Fachtagung der GI 1976.
Informatik Fachberichte 1, Springer
- [See,74] Seegmüller,G. : Einführung in die Systemprogrammierung
B.I. Reihe Informatik, 1974

- Smith, B.S. : Error Detection in Formal Languages
Journal of Computer and Systems
Sciences, Vol.4, Oct. 1970
- [Spr, 76] Springer, P. : Automatische Fehlerbehandlung in
Sprachübersetzern mit Hilfe einer
Nachsilbengrammatik
Diplomarbeit, 1976 TU München
- Souza, C.R., : Syntactical Decodes and Backtracking
Scholtz, R.A. Grammars
TR A69-9, Univ. of Hawaii,
Honolulu, 1963
- Souza, C.R., : Probabilistic Generation, Transmission
Scholtz, R.A. and Syntactic Decoding of Contextfree
Languages
Eight Annual Allerson Conference,
October, 1970
- Teitelbaum, R. : Context-Free Error Analysis by Evaluation
of Algebraic Power Series
Proc.ACM-SIGACT, Fifth Ann. Conf. on
Theory of Computing, Univ.of Texas,
Austin, Texas, 1973
- Younger, D.A. : Recognition and Parsing of Context-Free
Languages in Time n^3
Information and Control 10, Feb. 1967
- Wagner, R.A., : The String-to-String Correction
Fisher, M.J. Problem
JACM, Vol.21, No.1, January 1974

- Wagner,R.A. : Order - n Correction for Regular
Languages
CACM, Vol.17, No.5, May 1974
- Wetherell,Ch. : Why Automatic Error Correctors Fail
Computer Languages, Vol 2., 1977
- [Wir,68] Wirth,N. : PL360, A Programming Language for the
360 Computers
Journal of the ACM, Vol.15, No.1,
January 1968
- [Wir.Jen,74] Wirth,N., : PASCAL User Manual and Report
Jensen,K. Lecture Notes in Computer Science 18
Springer, 1974
- [Wir.Web,66] Wirth,N., : EULER : A Generalisation of Algol
Weber,H. and its Formal Definition: Part I

7. INDEX

7.1 Symbole

$\wedge$	logisches und	8
$\vee$	logisches oder	8
$\Leftrightarrow$	genau dann	10
$\rangle$	es folgt	10
$\forall$	für alle	10
$\exists$	es existiert	9
$:=$	definiert durch	9
$\{x_1, \dots, x_n\}$	Menge bestehend aus den Elementen $x_1, \dots, x_n$	8
$\{x \mid P(x)\}$	Menge bestehend aus den Elementen, welche die Eigenschaft $P(x)$ haben	8
$\in$	Elementbeziehung	8
$\notin$	Negation der Elementbeziehung	8
$\emptyset$	leere Menge	8
$ M $	Mächtigkeit der Menge M	11
$A \subseteq B$	A Teilmenge von B	9
$A \subset B$	A echte Teilmenge von B	9
$A \cup B$	Vereinigung der Mengen A und B	9
$A \cap B$	Durchschnitt der Mengen A und B	9
$A \setminus B$	Komplement der Menge B bzgl. A	9
2^A	Potenzmenge von A	9
$\bigcup_{i \in I} A_i$	Vereinigung der Mengen A_i	9

(a,b)	geordnetes Paar	9
$A \times B$	kartesisches Produkt der Mengen A und B	9
A^i	n-faches kartesisches Produkt der Menge A	9
$(x_1, \dots, x_n)$	Element von A^i	9
$x_1 \dots x_n$	Element von A^i	9
x^s	Spiegelbild	10
$a R b$	a und b genügen der Relation R	10
$a \in R(b)$	a und b genügen der Relation R	10
R^+	transitive Hülle von der Relation R	10
R^*	transitive und reflexive Hülle der Relation R	10
$F : A \rightarrow B$	Abbildung F von A nach B	11
$F(a) = b$	b ist Bild von a unter F	11
A^+	Menge der nichtleeren Worte über dem Alphabet A	12
A^*	Menge aller Worte über dem Alphabet A	12
ϵ	leere Zeichenfolge	13
$ \alpha $	Länge von α	12
$x \circ y$	Konkatenation von x mit y	12
$G = (N, \Sigma, P, S)$	kontextfreie Grammatik	14
KFG	kontextfreie Grammatik	14
N	Menge der nichtterminalen Zeichen	14
Σ	Menge der terminalen Zeichen	14
Γ_G	Alphabet der Grammatik G	14

P	Menge der Produktionen	14
$V \rightarrow \beta$	Produktion	14
$V \rightarrow \beta_1 \beta_2 \dots \beta_n$	steht für $V \rightarrow \beta_1$, $V \rightarrow \beta_2, \dots, V \rightarrow \beta_n$	14
$LS(V \rightarrow \beta)$	linke Seite	18
$RS(V \rightarrow \beta)$	rechte Seite	18
S	Startsymbol	14
L(G)	von G erzeugte Sprache	15
$\leftarrow, \overset{\cdot}{=}, \rightarrow$	Präzedenzrelationen	22
EPS	einfache Präzedenzsprache	23
$\#$	Randzeichen	23
A	Analysator	16
FM	Menge der Fehlermeldungen	16
(D, a, j)	Fehler	75
(I, a, j)	Fehler	76
$(R, (a, b), j)$	Fehler	76
$KS(x, y)$	Korrekturstelle	75
BU-Präfix	Bottom-Up Präfix	55
BU-Suffix	Bottom-Up Suffix	127
PPS	potentielles Primphrasensuffix	106
RS1, RS2, RS3, RS4	RS-Fehler	126
BUS1, BUS2, BUS3, BUS4	Fehlertyp	133

7.2 Spezielle Mengen

$\mathbb{N}$	Menge der natürlichen Zahlen	9
$\mathbb{N}^0$	$\mathbb{N} \cup \{0\}$	9
$\mathbb{Z}$	Menge der ganzen Zahlen	9
$\mathbb{N}_k$ bzw. $\mathbb{N}_k^0$	$\{1, \dots, k\}$ bzw. $\{0, 1, \dots, k\}$	12
$\text{Infix}(v)$	Menge der Infixe von v	13
$\text{Infix}(L)$	Menge der Infixe von L	13
$\text{Infix}(V \rightarrow \beta)$	Infix von $V \rightarrow \beta$	18
$\text{Präfix}(v)$	Menge der Präfixe von v	13
$\text{Präfix}(L)$	Menge der Präfixe von L	13
$\text{Präfix}(V \rightarrow \beta)$	Präfix von $V \rightarrow \beta$	18
$\text{Suffix}(v)$	Menge aller Suffixe von v	13
$\text{Suffix}(L)$	Menge aller Suffixe von L	13
$\text{Suffix}(V \rightarrow \beta)$	Suffix von $V \rightarrow \beta$	18
Akonf_G	Anfangskonfiguration	31
Zkonf_G	Zwischenkonfiguration	31
Ekonf_G	Endekonfiguration	31
Konf_G	Konfiguration	31
$\text{NS}_{L,a}$	Nachsilbe von a bzgl. L	100
NS_L	Nachsilbensprache von L	100
$L1$	Beispielsprache	19
$N1$	nichtterminales Alphabet von $G1$	21
$\Sigma 1$	terminales Alphabet von $G1$	21
$G1$	Grammatik von $L1$	21
$\text{Ges}(\# \gamma)$	Menge der gesuchten Variablen	59

$\overline{\text{Ges}(\# \gamma)}$	Menge der gesuchten Variablen	128
$\# \Sigma^*$	$\{\# \} \cdot \Sigma^*$	24
$\Sigma^* \#$	$\Sigma^* \cdot \{\# \}$	24
$\# \Sigma^* \#$	$\{\# \} \Sigma^* \{\# \}$	24
$\# \Gamma^*, \Gamma^* \#, \# \Gamma^* \#$	analog	24

7.3 Spezielle Relationen

$x = \text{Anf}(u)$	x ist Anfang von u	13
$X \text{ Anf } V$	X ist Anfang von V	18
$x = \text{End}(u)$	x ist Ende von u	13
$X \text{ End } V$	X ist Ende von V	18
$x \xrightarrow[G]{\Rightarrow} y$	x produziert direkt y (Relation)	14,24
$x \xrightarrow[G]{R} y$	x produziert direkt y durch Rechtsableitung	14,24
$x \xrightarrow[G]{\Rightarrow^*} y$	transitive Hülle von $x \xrightarrow[G]{\Rightarrow} y$	14
$x \xrightarrow[G]{R^*} y$	transitive und reflexive Hülle von $x \xrightarrow[G]{\Rightarrow} y$	14
$x \xrightarrow[G]{R^+} y$	transitive Hülle von $x \xrightarrow[G]{R} y$	14
$x \xrightarrow[G]{R^*} y$	transitive und reflexive Hülle von $x \xrightarrow[G]{R} y$	14
$x \text{ ? } y$	zwischen x und y besteht keine Präzedenzrelation	22
$x \text{ } \odot \text{ } y$	zwischen x und y besteht eine Präzedenzrelation	83

7.4 Spezielle Abbildungen

Num_A	Numerierung von A	12
F1	Analysefunktion	33
F2	Analysefunktion	46
F3	Analysefunktion	48
F4	Analysefunktion	68
A_F	von der Analysefunktion F induzierter Analysator	32
$I(x)$	elementarer Korrekturoperator	73
$D(x)$	elementarer Korrekturoperator	73
$R(x)$	elementarer Korrekturoperator	73
$C^n(x)$	Menge der Zeichenfolgen, die mit Hilfe von n Korrekturen aus x gewonnen werden	73
$C^n(L)$	Menge der Zeichenfolgen, die mit Hilfe von maximal n Korrekturen auf ein Element von L abgebildet werden können	73
$HA(x,y)$	Hammingabstand von x und y	74
$\text{Ham}_L(x)$	durch den Hammingabstand definierter Fehlerbegriff	74
NSA_L	Nachsilbenanalysator für L	100
$\overline{F1}$	Nachsilbenanalysefunktion	116
$\overline{F2}$	Nachsilbenanalysefunktion	123
$\overline{F3}$	Nachsilbenanalysefunktion	124
$\overline{F4}$	Nachsilbenanalysefunktion	132
NSA_F	von der Nachsilbenanalysefunktion F induzierter Nachsilbenanalysator	115

$\mathcal{M}_{A,NSA}$	von A und NSA induzierter fehlerbehandeln- der Analysator	101
$f1, g1$	Präzedenzfunktionen für L1	140
$\tilde{F1}$	Analysefunktion	141
$\tilde{F4}$	Analysefunktion	141
$\tilde{F1} - \tilde{F4}$	Nachsilbenanalysefunktionen	144

7.5 Wichtige Begriffe

Abbildung		
-	bijektive	11
-	injektive	11
-	partielle	11
-	surjektive	11
-	totale	11
Ableitung		15,16
-	Rechts-	16
ableitbar		14,15
-	direkt	14,15
-	nichttrivial	14,15
Alphabet		13,14
Analyse		32,114
-	Algorithmus	16
-	- fehlerbehandelnder	79
-	Funktion	33,114
Analysator		16
-	fehlerbehandelnder	79
Anfang		13,18
Durchschnitt		9
Elementbeziehung		8
Ende		13,18
Fehler		75
-	Begriff	72
-	Diagnose	76
-	Stelle	76
-	Kellerungs-	46
-	Reduktions-	34,117
-	RS-	49
-	Typ1-	34
-	Zeichenpaar-	34
-	Anzahl der -	74
-	Anzahl der (von A) gemeldeten -	79

Folge,	unendliche	11
-	endliche	11
Funktion		11
gesuchte Variable,	Menge der	57,128
gleichmächtig		11
Grammatik,	kontextfreie	14
Infix		13,18
Hammingabstand		74
Hülle,	transitive	10
-	transitive und reflexive	10
Kandidat		80
Kandidatenwahl		80
Kellerinhalt		31
Komplement		9
Kondensationsphase		89
Konfiguration		31
-	Anfangs-	31
-	Ende-	31
-	Zwischen-	31
-	durch F definierte -	32,115
Konkatenation		2
Korrektur		73,80
-	Folge (kanonische)	76
-	Methode	2
-	Operator, delementarer	73
-	Phase	90
-	Stelle (mögliche)	75
-	global minimale	2
-	lokale	2
-	lokal syntaktisch korrekte -	90
Kosten		84
Länge	(eines Wortes)	12
LP-Eigenschaft		43,100

Menge		8
-	Potenz	9
-	Teil-	8
-	abzählbare	11
-	endliche	11
-	überabzählbare	11
Mustervergleichs-Verfahren		91
Nachsilbe		100
Nachsilbenanalysator		100
Nachsilbenanalyse		95
-	Funktion	114
-	minimale	96
Nachsilbensprache		100
Numerierung		12
Paar, geordnetes		9
Präfix		13, 18
-	Darstellung	50, 54, 128
-	aktuelles	47, 50
-	Bottom-Up	51
-	längstes	42
Präzedenzfunktion		139
Präzedenzgrammatik, einfache		22, 23
Präzedenzrelation		22
Präzedenzsprache, einfache		23
Primphrase		28
Primphrasensuffix, potentiell		106
Produkt, k-faches (einer Relation)		10
-	kartesisches	9
Produktion		14
produziert		14, 15
-	direkt	14, 15
-	nichttrivial	14, 15
Reduktion (einer unvollständigen rechten Seite)		117
Relation		10
Relationenmatrix		23
Rückwärtsschritt, erster		89
-	zweiter	89

Satz		15
Satzform		16
-	rechtskanonische	16
Seite ,	linke	18
-	rechte	18
Sprache,	formale	13
-	erzeugte	15
Suffix		13, 18
-	Bottom-Up	127
Variable,	syntaktische	14
Vereinigung		9
Vorwärtsschritt		89
Wiederaufsetzmethode		2
-	kellergesteuerte	4
-	zeichengesteuerte	4
Wort		15
Zeichen,	aktuelles	31
-	nichtterminales	14
-	terminales	14
Zeichenfolge		13
-	beabsichtigte	74

7.6 Definitionen, Sätze, Hilfssätze, Bemerkungen, Beispiele, Bilder

Definitionen		Sätze		Hilfssätze		Bemerkungen		Beispiele		Bilder	
Nr.	Seite	Nr.	Seite	Nr.	Seite	Nr.	Seite	Nr.	Seite	Nr.	Seite
1.1	14	1.1	32	1.1	34	1.1	16	1.1	19	1.1	20
1.2	14	1.2	33	1.2	34	1.2	22	1.2	23	1.2	21
1.3	15					1.3	34	1.3	27	1.3	25
1.4	15					1.4	41	1.4	28		
1.5	16										
1.6	18										
1.7	22										
1.8	22										
1.9	23										
1.10	23										
1.11	28										
1.12	31										
1.13	32										
1.14	33										
2.1	42	2.1	46					2.1	44		
2.2	43	2.2	49					2.2	44		
2.3	50	2.3	50					2.3	45		
2.4	55	2.4	55					2.4	47		
2.5	56	2.5	56					2.5	51		
2.6	59	2.6	59					2.6	52		
		2.7	64					2.7	53		
		2.8	69					2.8	60		
								2.9	62		
								2.10	68		
3.1	72	3.1	97			3.1	76	3.1	77		
3.2	73	3.2	101			3.2	80	3.2	81		
3.3	74	3.3	101			3.3	117	3.3	83		
3.4	74	3.4	103			3.4	125	3.4	87		
3.5	75	3.5	103			3.5	133	3.5	90		
3.6	75	3.6	115					3.6	92		
3.7	79	3.7	116					3.7	94		
3.8	95	3.8	123					3.8	96		
3.9	100	3.9	124					3.9	98		
3.10	100	3.10	127					3.10	106		
3.11	101	3.11	128					3.11	108		
3.12	114	3.12	130					3.12	110		
3.13	114	3.13	131					3.13	112		
3.14	127	3.14	133					3.14	112		
3.15	128							3.15	121		
3.16	128							3.16	126		
								3.17	131		
								3.18	133		
								3.19	135		
4.1	139	4.1	141			4.1	139	4.1	142	4.1	140
		4.2	143								

LEBENS LAUF:

Geburtsdatum: 10. 8. 1944

Geburtsort: Freising

Schulbildung: Besuch der Volksschule in München,
von 1950 bis 1955
Besuch der Ludwig-Thoma Oberrealschule
in Prien am Chiemsee, von 1955 bis 1964;
Abitur 1964

Studium: Studium der Mathematik an der Ludwig-
Maximilians-Universität in München,
von 1964 bis 1970; Vordiplom 1967

Abschlußprüfungen: Hauptdiplom in Mathematik 1970

Berufliche Tätigkeiten: Seit 1970 bin ich als wissenschaftlicher
Mitarbeiter am Leibniz-Rechenzentrum der
Bayerischen Akademie der Wissenschaften
in München angestellt.

